

EV355227078

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Programming Interface for a Computer Platform

Inventor(s):

Rob Relyea

Jeff Bogdan

ATTORNEY'S DOCKET NO. MS1-1779US

TECHNICAL FIELD

This invention relates to software and to development of such software. More particularly, this invention relates to a programming interface that facilitates use of a software platform by application programs and computer hardware.

BRIEF DESCRIPTION OF ACCOMPANYING COMPACT DISCS

Accompanying this specification is a set of three compact discs that stores a Software Development Kit (SDK) for the Microsoft® Windows® Code-Named "Longhorn" operating system. The SDK contains documentation for the Microsoft® Windows® Code-Named "Longhorn" operating system. Duplicate copies of each of these three compact discs also accompany this specification.

The first compact disc in the set of three compact discs (CD 1 of 3) includes a file folder named "lhsdk" that was created on October 22, 2003; it is 586 Mbytes in size, contains 9,692 sub-folders, and contains 44,292 sub-files. The second compact disc in the set of three compact discs (CD 2 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 605 Mbytes in size, contains 12,628 sub-folders, and contains 44,934 sub-files. The third compact disc in the set of three compact discs (CD 3 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 575 Mbytes in size, contains 9,881 sub-folders, and contains 43,630 sub-files. The files on each of these three compact discs can be executed on a Windows®-based computing device (e.g., IBM-PC, or equivalent) that executes a Windows®-brand operating system (e.g., Windows® NT, Windows® 98, Windows® 2000, Windows® XP, etc.). The files on each compact disc in this set of three compact discs are hereby incorporated by reference.

1 Each compact disc in the set of three compact discs itself is a CD-R, and
2 conforms to the ISO 9660 standard. The contents of each compact disc in the set
3 of three compact discs is in compliance with the American Standard Code for
4 Information Interchange (ASCII).

5 6 **BACKGROUND**

7 Very early on, computer software came to be categorized as “operating
8 system” software or “application” software. Broadly speaking, an application is
9 software meant to perform a specific task for the computer user such as solving a
10 mathematical equation or supporting word processing. The operating system is
11 the software that manages and controls the computer hardware. The goal of the
12 operating system is to make the computer resources available to the application
13 programmer while at the same time, hiding the complexity necessary to actually
14 control the hardware.

15 The operating system makes the resources available via functions that are
16 collectively known as the Application Program Interface or API. The term API is
17 also used in reference to a single one of these functions. The functions are often
18 grouped in terms of what resource or service they provide to the application
19 programmer. Application software requests resources by calling individual API
20 functions. API functions also serve as the means by which messages and
21 information provided by the operating system are relayed back to the application
22 software.

23 In addition to changes in hardware, another factor driving the evolution of
24 operating system software has been the desire to simplify and speed application
25 software development. Application software development can be a daunting task,

1 sometimes requiring years of developer time to create a sophisticated program
2 with millions of lines of code. For a popular operating system such as various
3 versions of the Microsoft Windows® operating system, application software
4 developers write thousands of different applications each year that utilize the
5 operating system. A coherent and usable operating system base is required to
6 support so many diverse application developers.

7 Often, development of application software can be made simpler by making
8 the operating system more complex. That is, if a function may be useful to several
9 different application programs, it may be better to write it once for inclusion in the
10 operating system, than requiring dozens of software developers to write it dozens
11 of times for inclusion in dozens of different applications. In this manner, if the
12 operating system supports a wide range of common functionality required by a
13 number of applications, significant savings in applications software development
14 costs and time can be achieved.

15 Regardless of where the line between operating system and application
16 software is drawn, it is clear that for a useful operating system, the API between
17 the operating system and the computer hardware and application software is as
18 important as efficient internal operation of the operating system itself.

19 Over the past few years, the universal adoption of the Internet, and
20 networking technology in general, has changed the landscape for computer
21 software developers. Traditionally, software developers focused on single-site
22 software applications for standalone desktop computers, or LAN-based computers
23 that were connected to a limited number of other computers via a local area
24 network (LAN). Such software applications were typically referred to as “shrink
25 wrapped” products because the software was marketed and sold in a shrink-

1 wrapped package. The applications utilized well-defined APIs to access the
2 underlying operating system of the computer.

3 As the Internet evolved and gained widespread acceptance, the industry
4 began to recognize the power of hosting applications at various sites on the World
5 Wide Web (or simply the "Web"). In the networked world, clients from anywhere
6 could submit requests to server-based applications hosted at diverse locations and
7 receive responses back in fractions of a second. These Web applications, however,
8 were typically developed using the same operating system platform that was
9 originally developed for standalone computing machines or locally networked
10 computers. Unfortunately, in some instances, these applications do not adequately
11 transfer to the distributed computing regime. The underlying platform was simply
12 not constructed with the idea of supporting limitless numbers of interconnected
13 computers.

14 To accommodate the shift to the distributed computing environment being
15 ushered in by the Internet, Microsoft Corporation developed a network software
16 platform known as the ".NET" Framework (read as "Dot Net"). Microsoft® .NET
17 is software for connecting people, information, systems, and devices. The
18 platform allows developers to create Web services that will execute over the
19 Internet. This dynamic shift was accompanied by a set of API functions for
20 Microsoft's .NET™ Framework.

21 As use of the .NET™ Framework has become increasingly common, ways
22 to increase the efficiency and/or performance of the platform have been identified.
23 The inventors have developed a unique set of API functions to allow for such
24 increased efficiency and/or performance.
25

SUMMARY

A programming interface provides functions for generating applications, documents, media presentations and other content. These functions allow developers to obtain services from an operating system, object model service, or other system or service. In one embodiment, the functions allow a developer to generate a graphical user interface.

BRIEF DESCRIPTION OF THE DRAWINGS

The same numbers are used throughout the drawings to reference like features.

Fig. 1 illustrates a network architecture in which clients access Web services over the Internet using conventional protocols.

Fig. 2 is a block diagram of a software architecture for a network platform, which includes an application program interface (API).

Fig. 3 is a block diagram of the presentation subsystem supported by the API, as well as function classes of the various API functions.

Fig. 4 is a block diagram of an exemplary computer that may execute all or part of the software architecture.

Figs. 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 and 16 illustrate various example implementations of a programming interface.

DETAILED DESCRIPTION

This disclosure addresses an application program interface (API) for a network platform upon which developers can build Web applications and services. More particularly, an exemplary API is described for operating systems that make

1 use of a network platform, such as the .NET™ Framework created by Microsoft
2 Corporation. The .NET™ Framework is a software platform for Web services and
3 Web applications implemented in the distributed computing environment. It
4 represents the next generation of Internet computing, using open communication
5 standards to communicate among loosely coupled Web services that are
6 collaborating to perform a particular task.

7 In the described implementation, the network platform utilizes XML
8 (extensible markup language), an open standard for describing data. XML is
9 managed by the World Wide Web Consortium (W3C). XML is used for defining
10 data elements on a Web page and business-to-business documents. XML uses a
11 similar tag structure as HTML; however, whereas HTML defines how elements
12 are displayed, XML defines what those elements contain. HTML uses predefined
13 tags, but XML allows tags to be defined by the developer of the page. Thus,
14 virtually any data items can be identified, allowing Web pages to function like
15 database records. Through the use of XML and other open protocols, such as
16 Simple Object Access Protocol (SOAP), the network platform allows integration
17 of a wide range of services that can be tailored to the needs of the user. Although
18 the embodiments described herein are described in conjunction with XML and
19 other open standards, such are not required for the operation of the claimed
20 invention.

21 As used herein, the phrase application program interface or API includes
22 traditional interfaces that employ method or function calls, as well as remote calls
23 (e.g., a proxy, stub relationship) and SOAP/XML invocations.

24 It should be appreciated that in some of namespace descriptions below,
25 descriptions of certain classes, interfaces, enumerations and delegates are left

1 blank. More complete descriptions of these classes, interfaces, enumerations and
2 delegates can be found in the subject matter of the compact discs that store the
3 SDK referenced above.

4 5 EXEMPLARY NETWORK ENVIRONMENT

6 Fig. 1 shows a network environment 100 in which a network platform, such
7 as the .NET™ Framework, may be implemented. The network environment 100
8 includes representative Web services 102(1), ..., 102(N), which provide services
9 that can be accessed over a network 104 (e.g., Internet). The Web services,
10 referenced generally as number 102, are programmable application components
11 that are reusable and interact programmatically over the network 104, typically
12 through industry standard Web protocols, such as XML, SOAP, WAP (wireless
13 application protocol), HTTP (hypertext transport protocol), and SMTP (simple
14 mail transfer protocol) although other means of interacting with the Web services
15 over the network may also be used, such as Remote Procedure Call (RPC) or
16 object broker type technology. A Web service can be self-describing and is often
17 defined in terms of formats and ordering of messages.

18 Web services 102 are accessible directly by other services (as represented
19 by communication link 106) or a software application, such as Web application
20 110 (as represented by communication links 112 and 114). Each Web service 102
21 is illustrated as including one or more servers that execute software to handle
22 requests for particular services. Such services often maintain databases that store
23 information to be served back to requesters. Web services may be configured to
24 perform any one of a variety of different services. Examples of Web services
25 include login verification, notification, database storage, stock quoting, location

1 directories, mapping, music, electronic wallet, calendar/scheduler, telephone
2 listings, news and information, games, ticketing, and so on. The Web services can
3 be combined with each other and with other applications to build intelligent
4 interactive experiences.

5 The network environment 100 also includes representative client devices
6 120(1), 120(2), 120(3), 120(4), ..., 120(M) that utilize the Web services 102 (as
7 represented by communication link 122) and/or the Web application 110 (as
8 represented by communication links 124, 126, and 128). The clients may
9 communicate with one another using standard protocols as well, as represented by
10 an exemplary XML link 130 between clients 120(3) and 120(4).

11 The client devices, referenced generally as number 120, can be
12 implemented many different ways. Examples of possible client implementations
13 include, without limitation, portable computers, stationary computers, tablet PCs,
14 televisions/set-top boxes, wireless communication devices, personal digital
15 assistants, gaming consoles, printers, photocopiers, and other smart devices.

16 The Web application 110 is an application designed to run on the network
17 platform and may utilize the Web services 102 when handling and servicing
18 requests from clients 120. The Web application 110 is composed of one or more
19 software applications 130 that run atop a programming framework 132, which are
20 executing on one or more servers 134 or other computer systems. Note that a
21 portion of Web application 110 may actually reside on one or more of clients 120.
22 Alternatively, Web application 110 may coordinate with other software on clients
23 120 to actually accomplish its tasks.

24 The programming framework 132 is the structure that supports the
25 applications and services developed by application developers. It permits multi-

1 language development and seamless integration by supporting multiple languages.
2 It supports open protocols, such as SOAP, and encapsulates the underlying
3 operating system and object model services. The framework provides a robust and
4 secure execution environment for the multiple programming languages and offers
5 secure, integrated class libraries.

6 The framework 132 is a multi-tiered architecture that includes an
7 application program interface (API) layer 142, a common language runtime (CLR)
8 layer 144, and an operating system/services layer 146. This layered architecture
9 allows updates and modifications to various layers without impacting other
10 portions of the framework. A common language specification (CLS) 140 allows
11 designers of various languages to write code that is able to access underlying
12 library functionality. The specification 140 functions as a contract between
13 language designers and library designers that can be used to promote language
14 interoperability. By adhering to the CLS, libraries written in one language can be
15 directly accessible to code modules written in other languages to achieve seamless
16 integration between code modules written in one language and code modules
17 written in another language. One exemplary detailed implementation of a CLS is
18 described in an ECMA standard created by participants in ECMA TC39/TG3.
19 The reader is directed to the ECMA web site at www.ecma.ch.

20 The API layer 142 presents groups of functions that the applications 130
21 can call to access the resources and services provided by layer 146. By exposing
22 the API functions for a network platform, application developers can create Web
23 applications for distributed computing systems that make full use of the network
24 resources and other Web services, without needing to understand the complex
25 interworkings of how those network resources actually operate or are made

1 available. Moreover, the Web applications can be written in any number of
2 programming languages, and translated into an intermediate language supported
3 by the common language runtime 144 and included as part of the common
4 language specification 140. In this way, the API layer 142 can provide methods
5 for a wide and diverse variety of applications.

6 Additionally, the framework 132 can be configured to support API calls
7 placed by remote applications executing remotely from the servers 134 that host
8 the framework. Representative applications 148(1) and 148(2) residing on clients
9 120(3) and 120(M), respectively, can use the API functions by making calls
10 directly, or indirectly, to the API layer 142 over the network 104.

11 The framework may also be implemented at the clients. Client 120(3)
12 represents the situation where a framework 150 is implemented at the client. This
13 framework may be identical to server-based framework 132, or modified for client
14 purposes. Alternatively, the client-based framework may be condensed in the
15 event that the client is a limited or dedicated function device, such as a cellular
16 phone, personal digital assistant, handheld computer, or other
17 communication/computing device.

18 19 DEVELOPERS' PROGRAMMING FRAMEWORK

20 Fig. 2 shows the programming framework 132 in more detail. The
21 common language specification (CLS) layer 140 supports applications written in a
22 variety of languages 130(1), 130(2), 130(3), 130(4), ..., 130(K). Such application
23 languages include Visual Basic, C++, C#, COBOL, Jscript, Perl, Eiffel, Python,
24 and so on. The common language specification 140 specifies a subset of features
25 or rules about features that, if followed, allow the various languages to

1 communicate. For example, some languages do not support a given type (e.g., an
2 “int*” type) that might otherwise be supported by the common language runtime
3 144. In this case, the common language specification 140 does not include the
4 type. On the other hand, types that are supported by all or most languages (e.g.,
5 the “int[]” type) is included in common language specification 140 so library
6 developers are free to use it and are assured that the languages can handle it. This
7 ability to communicate results in seamless integration between code modules
8 written in one language and code modules written in another language. Since
9 different languages are particularly well suited to particular tasks, the seamless
10 integration between languages allows a developer to select a particular language
11 for a particular code module with the ability to use that code module with modules
12 written in different languages. The common language runtime 144 allow seamless
13 multi-language development, with cross language inheritance, and provide a
14 robust and secure execution environment for the multiple programming languages.
15 For more information on the common language specification 140 and the common
16 language runtime 144, the reader is directed to co-pending applications entitled
17 “Method and System for Compiling Multiple Languages”, filed 6/21/2000 (serial
18 number 09/598,105) and “Unified Data Type System and Method” filed 7/10/2000
19 (serial number 09/613,289), which are incorporated by reference.

20 The framework 132 encapsulates the operating system 146(1) (e.g.,
21 Windows®-brand operating systems) and object model services 146(2) (e.g.,
22 Component Object Model (COM) or Distributed COM). The operating system
23 146(1) provides conventional functions, such as file management, notification,
24 event handling, user interfaces (e.g., windowing, menus, dialogs, etc.), security,
25 authentication, verification, processes and threads, memory management, and so

1 on. The object model services 146(2) provide interfacing with other objects to
2 perform various tasks. Calls made to the API layer 142 are handed to the common
3 language runtime layer 144 for local execution by the operating system 146(1)
4 and/or object model services 146(2).

5 The API 142 groups API functions into multiple namespaces. Namespaces
6 essentially define a collection of classes, interfaces, delegates, enumerations, and
7 structures, which are collectively called “types”, that provide a specific set of
8 related functionality. A class represents managed heap allocated data that has
9 reference assignment semantics. A delegate is an object oriented function pointer.
10 An enumeration is a special kind of value type that represents named constants. A
11 structure represents static allocated data that has value assignment semantics. An
12 interface defines a contract that other types can implement.

13 By using namespaces, a designer can organize a set of types into a
14 hierarchical namespace. The designer is able to create multiple groups from the
15 set of types, with each group containing at least one type that exposes logically
16 related functionality. In the exemplary implementation, the API 142 is organized
17 to include three root namespaces. It should be noted that although only three root
18 namespaces are illustrated in Fig. 2, additional root namespaces may also be
19 included in API 142. The three root namespaces illustrated in API 142 are: a first
20 namespace 200 for a presentation subsystem (which includes a namespace 202 for
21 a user interface shell), a second namespace 204 for web services, and a third
22 namespace 206 for a file system. Each group can then be assigned a name. For
23 instance, types in the presentation subsystem namespace 200 can be assigned the
24 name “Windows”, and types in the file system namespace 206 can be assigned
25 names “Storage”. The named groups can be organized under a single “global

1 root” namespace for system level APIs, such as an overall System namespace. By
2 selecting and prefixing a top level identifier, the types in each group can be easily
3 referenced by a hierarchical name that includes the selected top level identifier
4 prefixed to the name of the group containing the type. For instance, types in the
5 file system namespace 206 can be referenced using the hierarchical name
6 “System.Storage”. In this way, the individual namespaces 200, 204, and 206
7 become major branches off of the System namespace and can carry a designation
8 where the individual namespaces are prefixed with a designator, such as a
9 “System.” prefix.

10 The presentation subsystem namespace 200 pertains to programming and
11 content development. It supplies types that allow for the generation of
12 applications, documents, media presentations and other content. For example,
13 presentation subsystem namespace 200 provides a programming model that allows
14 developers to obtain services from the operating system 146(1) and/or object
15 model services 146(2).

16 The shell namespace 202 pertains to user interface functionality. It supplies
17 types that allow developers to embed user interface functionality in their
18 applications, and further allows developers to extend the user interface
19 functionality.

20 The web services namespace 204 pertains to an infrastructure for enabling
21 creation of a wide variety of applications, e.g. applications as simple as a chat
22 application that operates between two peers on an intranet, and/or as complex as a
23 scalable Web service for millions of users. The described infrastructure is
24 advantageously highly variable in that one need only use those parts that are
25 appropriate to the complexity of a particular solution. The infrastructure provides

1 a foundation for building message-based applications of various scale and
2 complexity. The infrastructure or framework provides APIs for basic messaging,
3 secure messaging, reliable messaging and transacted messaging. In the
4 embodiment described below, the associated APIs have been factored into a
5 hierarchy of namespaces in a manner that has been carefully crafted to balance
6 utility, usability, extensibility and versionability.

7 The file system namespace 206 pertains to storage. It supplies types that
8 allow for information storage and retrieval.

9 In addition to the framework 132, programming tools 210 are provided to
10 assist the developer in building Web services and/or applications. One example of
11 the programming tools 210 is Visual Studio™, a multi-language suite of
12 programming tools offered by Microsoft Corporation.

13 14 ROOT API NAMESPACES

15 Fig. 3 shows a portion of the presentation subsystem 200 in more detail. In
16 one embodiment, the namespaces are identified according to a hierarchical naming
17 convention in which strings of names are concatenated with periods. For instance,
18 the presentation subsystem namespace 200 is identified by the root name
19 “System.Windows”. Within the “System.Windows” namespace is another
20 namespace for various controls, identified as “System.Windows.Controls”, which
21 further identifies another namespace for primitives (not shown) known as
22 “System.Windows.Controls.Primitives”. With this naming convention in mind,
23 the following provides a general overview of selected namespaces of the API 142,
24 although other naming conventions could be used with equal effect.
25

1 As shown in Fig. 3, the presentation subsystem 200 includes multiple
2 namespaces. The namespaces shown in Fig. 3 represent a particular embodiment
3 of the presentation subsystem 200. Other embodiments of the presentation
4 subsystem 200 may include one or more additional namespaces or may omit one
5 or more of the namespaces shown in Fig. 3.

6 The presentation subsystem 200 is the root namespace for much of the
7 presentation functionality of the API 142. A controls namespace 310 includes
8 controls used to build a display of information, such as a user interface, and
9 classes that allow a user to interact with an application. Example controls include
10 "Button" that creates a button on the display, "RadioButton" that generates a
11 radio-style button on the display, "Menu" that creates a menu on the display,
12 "ToolBar" that creates a toolbar on the display, "Image" that generates an image
13 on the display and "TreeView" that creates a hierarchical view of information.

14 Certain controls are created by nesting and arranging multiple elements.
15 The controls have a logical model that hides the elements used to create the
16 controls, thereby simplifying the programming model. The controls can be styled
17 and themed by a developer or a user (e.g., by customizing the appearance and
18 behavior of user interface buttons). Some controls have addressable components
19 that allow an individual to adjust the style of individual controls. Additionally, the
20 controls can be sub-classed and extended by application developers and
21 component developers. The controls are rendered using vector graphics such that
22 they can be resized to fit the requirements of a particular interface or other display.
23 The controls are capable of utilizing animation to enhance, for example, the
24 interactive feel of a user interface and to show actions and reactions.

Controls Namespace

The controls namespace 310 includes one or more panels, which are controls that measure and arrange their children (e.g., nested elements). For example, a “DockPanel” panel arranges children by docking each child to the top, left, bottom or right side of the display, and fills-in the remaining space with other data. For example, a particular panel may dock menus and toolbars to the top of the display, a status bar to the bottom of the display, a folder list to the left side of the display, and fills the rest of the space with a list of messages.

As mentioned above, System.Windows.Controls.Primitives is a namespace that includes multiple controls that are components typically used by developers of the controls in the System.Windows.Controls namespace and by developers creating their own controls. Examples of these components include “Thumb and RepeatButton”. “ScrollBar”, another component, is created using four repeat buttons (one for “line up”, one for “line down”, one for “page up”, and one for “page down”) and a “Thumb” for dragging the current view to another location in the document. In another example, “ScrollViewer” is a control created using two “ScrollBars” and one “ScrollArea” to provide a scrollable area.

The following list contains example classes exposed by the System.Windows.Controls namespace. These classes allow a user to interact with, for example, an application through various input and output capabilities as well as additional display capabilities.

- AccessKey – AccessKey is a FrameworkElement element that wraps a character, indicating that it is to receive keyboard cue decorations denoting

1 the character as a keyboard mnemonic. By default, the keyboard cue
2 decoration is an underline.

- 3 • Audio – Audio Element.
- 4
- 5 • Border – Draws a border, background, or both around another element.
- 6
- 7 • Button – Represents the standard button component that inherently reacts to
8 the Click event.
- 9
- 10 • Canvas – Defines an area within which a user can explicitly position child
11 elements by coordinates relative to the Canvas area.
- 12
- 13 • CheckBox – Use a CheckBox to give the user an option, such as true/false.
14 CheckBox allows the user to choose from a list of options. CheckBox
15 controls let the user pick a combination of options.
- 16
- 17 • CheckedChangedEventArgs – This CheckedChangedEventArgs class
18 contains additional information about the CheckedChangedEvent event.
- 19
- 20 • CheckStateChangedEventArgs – This CheckStateChangedEventArgs class
21 contains additional information about the CheckStateChangedEvent event.
- 22
- 23 • ClickEventArgs – Contains information about the Click event.
- 24
- 25 • ColumnStyle – Represents a changeable ColumnStyle object.
- ColumnStyles – Changeable pattern IList object that is a collection of
Changeable elements.
- ComboBox – ComboBox control.

1 ComboBoxItem – Control that implements a selectable item inside a
2 ComboBox.

- 3 • ContactPickerDialog – Allows a user to select one or more contacts.
- 4 • ContactPropertyRequest – Allows an application to request information
5 about a contact property through a ContactPickerDialog. This class cannot
6 be inherited.
- 7 • ContactPropertyRequest Collection – Represents a collection of
8 ContactPropertyRequest objects.
- 9 • ContactSelection – Information about a selected contact from Microsoft®
10 Windows® File System, code-named "WinFS" or Microsoft Active
11 Directory®.
- 12 • ContactSelectionCollection – Represents a collection of ContactSelection
13 objects.
- 14 • ContactTextBox – An edit control that supports picking contacts or
15 properties of contacts.
- 16 • ContactTextBoxSelectionChangedEventArgs – Arguments for the
17 ContactTextBoxSelectionChanged event.
- 18 • ContactTextBoxTextChangedEventArgs – Arguments for the
19 ContactTextBoxTextChanged event.
- 20 • ContactTextBoxTextResolvedEventArgs – Arguments for the
21 TextResolvedToContact event.
- 22 • ContactTextBoxTextResolvedEventArgs – Arguments for the
23 TextResolvedToContact event.
- 24 • ContactTextBoxTextResolvedEventArgs – Arguments for the
25 TextResolvedToContact event.

1 ContentChangedEventArgs – The event arguments for
2 ContentChangedEvent.

- 3 • ContentControl – The base class for all controls with a single piece of
4 content.
- 5 • ContentPresenter – ContentPresenter is used within the style of a content
6 control to denote the place in the control's visual tree (chrome template)
7 where the content is to be added.
- 8 • ContextMenu – Control that defines a menu of choices for users to invoke.
- 9 • ContextMenuEventArgs – The data sent on a ContextMenuEvent.
- 10 • Control – Represents the base class for all user-interactive elements. This
11 class provides a base set of properties for its subclasses.
- 12 • Decorator – Base class for elements that apply effects onto or around a
13 single child element, such as Border.
- 14 • DockPanel – Defines an area within which you can arrange child elements
15 either horizontally or vertically, relative to each other.
- 16 • DragDeltaEventArgs – This DragDeltaEventArgs class contains additional
17 information about the DragDeltaEvent event.
- 18 • FixedPanel – FixedPanel is the root element used in fixed-format
19 documents to contain fixed pages for pagination. FixedPanel displays
20 paginated content one page at a time or as a scrollable stack of pages.
- 21 • FixedPanel – FixedPanel is the root element used in fixed-format
22 documents to contain fixed pages for pagination. FixedPanel displays
23 paginated content one page at a time or as a scrollable stack of pages.
- 24 • FixedPanel – FixedPanel is the root element used in fixed-format
25 documents to contain fixed pages for pagination. FixedPanel displays
26 paginated content one page at a time or as a scrollable stack of pages.

1 FlowPanel – FlowPanel is used to break, wrap, and align content that
2 exceeds the length of a single line. FlowPanel provides line-breaking and
3 alignment properties that can be used when the flow of the container's
4 content, Text for example, is likely to exceed the length of a single line.

- 5 • Frame – An area that can load the contents of another markup tree.
- 6
- 7 • Generator – Generator is the object that generates a UI on behalf of an
8 ItemsControl, working under the supervision of a GeneratorFactory.
- 9 • GeneratorFactory – A GeneratorFactory is responsible for generating the UI
10 on behalf of an ItemsControl. It maintains the association between the
11 items in the control's ItemsCollection (flattened view) and the
12 corresponding UIElements. The control's item-container can ask the
13 factory for a Generator, which does the actual generation of UI.
- 14
- 15 • GridPanel – Defines a grid area consisting of columns and rows.
- 16 • HeaderItemsControl – The base class for all controls that contain multiple
17 items and have a header.
- 18 • HorizontalScrollBar – The Horizontal ScrollBar class.
- 19
- 20 • HorizontalSlider – The Horizontal Slider class.
- 21 • HyperLink – The HyperLink class implements navigation control. The
22 default presenter is TextPresenter.
- 23
- 24 • Image – Provides an easy way to include an image in a document or an
25 application.

1 IncludeContactEventArgs – Arguments passed to handlers of the
2 ContactPickerDialog.IncludeContact event.

- 3 • ItemCollection – Maintains a collection of discrete items within a control.
4 Provides methods and properties that enable changing the collection
5 contents and obtaining data about the contents.
6
- 7 • ItemsChangedEventArgs – The ItemsChanged event is raised by a
8 GeneratorFactory to inform layouts that the items collection has changed.
- 9 • ItemsControl – The base class for all controls that have multiple children.
- 10 • ItemsView – ItemsView provides a flattened view of an ItemCollection.
- 11
- 12 • KeyboardNavigation – KeyboardNavigation class provide methods for
13 logical (Tab) and directional (arrow) navigation between focusable
14 controls.
- 15 • ListBox – Control that implements a list of selectable items.
- 16
- 17 • ListItem – Control that implements a selectable item inside a ListBox.
- 18
- 19 • Menu – Control that defines a menu of choices for users to invoke.
- 20
- 21 • MenuItem – A child item of Menu. MenuItems can be selected to invoke
22 commands. MenuItems can be separators. MenuItems can be headers for
23 submenus. MenuItems can be checked or unchecked.
- 24 • PageViewer – Represents a document-viewing composite control that
25 contains a pagination control, a toolbar, and a page bar control.

1 `PaginationCompleteEventArgs` – The event arguments for the
2 `PaginationCompleteEvent`.

- 3 • `PaginationProgressEventArgs` – The event arguments for the
4 `PaginationProgressEvent`.
- 5 • `Pane` – Provides a way to define window properties in a markup language
6 (e.g., "XAML") without launching a new window.
- 7 • `Panel` – Provides a base class for all `Panel` elements. In order to instantiate
8 a `Panel` element, use the derived concrete class.
- 9 • `RadioButton` – `RadioButton` implements an option button with two states:
10 true or false.
- 11 • `RadioButtonList` – This control serves as a grouping control for
12 `RadioButtons` and is the piece that handles `RadioButton` mutual exclusivity.
13 The `RadioButtonList` inherits from `Selector`. The `RadioButtonList` is
14 essentially a `Single SelectionMode Selector` and the concept of `Selection`
15 (from `Selector`) is keyed off of the `Checked` property of the `RadioButton` it
16 is grouping.
- 17 • `RowStyle` – Changeable pattern Changeable elements.
- 18 • `RowStyles` – Changeable pattern `ICollection` object that is a collection of
19 Changeable elements.
- 20 • `ScrollChangeEventArgs` – The `ScrollChangeEventArgs` describe a change
21 in scrolling state.
- 22 • `ScrollChangeEventArgs` – The `ScrollChangeEventArgs` describe a change
23 in scrolling state.
- 24 • `ScrollChangeEventArgs` – The `ScrollChangeEventArgs` describe a change
25 in scrolling state.

1 ScrollViewer –

2 SelectedItemsCollection – A container for the selected items in a Selector.

- 3
- 4 • SelectionChangedEventArgs – The inputs to a selection changed event
5 handler.
 - 6 • SimpleText – SimpleText is a lightweight, multi-line, single-format text
7 element intended for use in user interface (UI) scenarios. SimpleText
8 exposes several of the same formatting properties as Text and can often be
9 used for a performance gain at the cost of some versatility.
 - 10 • StyleSelector – StyleSelector allows the app writer to provide custom style
11 selection logic. For example, with a class Bug as the Content, use a
12 particular style for Pri1 bugs and a different style for Pri2 bugs. An
13 application writer can override the SelectStyle method in a derived selector
14 class and assign an instance of this class to the StyleSelector property on
15 ContentPresenter class.
 - 16
 - 17 • Text – Represents a Text control that enables rendering of multiple formats
18 of Text. Text is best used within an application UI; more advanced text
19 scenarios benefit from the additional feature set of TextPanel. In most
20 cases where relatively simple text support is required, Text is the preferred
21 element because of its lightweight nature and range of features.
 - 22 • TextBox – Represents the control that provides an editable region that
23 accepts text input.
 - 24
 - 25

- 1 • TextChangedEventArgs – The TextChangedEventArgs class represents a
2 type of RoutedEventArgs that are relevant to events raised by
3 TextRange.SetText().
- 4 • TextPanel – Formats, sizes, and draws text. TextPanel supports multiple
5 lines of text and multiple text formats.
- 6 • ToolTip – A control to display information when the user hovers over a
7 control.
- 8 • ToolTipEventArgs – The data sent on a ToolTipEvent.
- 9 • TransformDecorator – TransformDecorator contains a child and applies a
10 specified transform to it. TransformDecorator implements logic to measure
11 and arrange the child in its local (pre-transform) coordinates such that after
12 the transform, the child fits tightly within the decorator's space and uses
13 maximal area. The child therefore needs to have no knowledge that a
14 transform has been applied to it.
- 15 • UIElementCollection – A UIElementCollection is a ordered collection of
16 UIElements.
- 17 • ValueChangedEventArgs – This ValueChangedEventArgs class contains
18 additional information about the ValueChangedEvent event.
- 19 • VerticalScrollBar – The Vertical ScrollBar class.
- 20 • VerticalSlider – The Vertical Slider class.
- 21
- 22
- 23
- 24
- 25

1 Video – Plays a streaming video or audio file in a specified rectangle within
2 the current user coordinate system.

- 3 • VisibleChangedEventArgs – The VisibleChangedEventArgs class contains
4 additional information about the VisibleChangedEvent event.

5
6
7 The System.Windows.Controls namespace also contains various
8 enumerations. The following list contains example enumerations associated with
9 the System.Windows.Controls namespace.

- 10 • CharacterCase – Specifies the case of characters in a TextBox control when
11 the text is typed.
- 12
13 • CheckState – Specifies the state of a control, such as a check box, that can
14 be checked, unchecked, or set to an indeterminate state.
- 15 • ClickMode – Specifies when the Click event should fire.
- 16
17 • ContactControlPropertyPosition – Controls the position and display of the
18 property of the contact.
- 19 • ContactPickerDialogLayout – Specifies how the ContactPickerDialog
20 should display selected properties.
- 21 • ContactPropertyCategory – Specifies which value to treat as the default in
22 the case where a property has multiple values for the user could choose
23 from. For example, if "Work" is specified as the preferred category when
24 requesting a phone number property from the ContactPickerDialog, and the
25

1 user selects a contact with both a work and home phone number, the work
2 phone number appears as the default selection. The user can then use the
3 UI to choose the home phone number instead.

- 4 • **ContactPropertyType** – Specifies a property of a contact that the
5 **ContactPickerDialog** can ask the user for.
- 6
- 7 • **ContactType** – Specifies which contact types to display in the
8 **ContactPickerDialog**.
- 9 • **Direction** – This enumeration is used by the **GeneratorFactory** and
10 **Generator** to specify the direction in which the generator produces UI.
- 11
- 12 • **Dock** – Specifies the Dock position of a child element within a **DockPanel**.
- 13 • **GeneratorStatus** – This enumeration is used by the **GeneratorFactory** to
14 indicate its status.
- 15 • **KeyNavigationMode** – The type of **TabNavigation** property specify how
16 the container will move the focus when Tab navigation occurs.
- 17
- 18 • **MenuItemBehavior** – Defines the different behaviors that a **MenuItem**
19 could have.
- 20 • **MenuItemType** – Defines the different placement types of **MenuItems**.
- 21
- 22 • **Orientation** – Slider orientation types.
- 23 • **PageViewerFit** – Selects how pages should be fit into the **PageViewer's**
24 **Client area**.
- 25

1 PageViewerMode – Selects the current PageViewer mode Reflected in the
2 mode dropdown.

- 3 • ScrollerVisibility – ScrollerVisiblty defines the visiblity behavior of a
4 scrollbar.
- 5 • SelectionMode – Specifies the selection behavior for the ListBox.

6
7
8 “Position” is an example structure associated with the
9 System.Windows.Controls namespace. A user of the Generator describes
10 positions using this structure. For example: To start generating forward from the
11 beginning of the item list, specify position (-1, 0) and direction Forward. To start
12 generating backward from the end of the list, specify position (-1, 0) and direction
13 Backward. To generate the items after the element with index k, specify position
14 (k, 0) and direction Forward.

15
16
17 The following list contains example delegates associated with the
18 System.Windows.Controls namespace.

- 19 • CheckedChangedEventHandler – This delegate is used by handlers of the
20 CheckedChangedEvent event.
- 21 • CheckStateChangedEventHandler – This delegate is used by handlers of
22 the CheckStateChangedEvent event.

23
24 ClickEventHandler – Represents the methods that handle the Click event.

1 ContactTextBoxSelectionChangedEventHandler – A delegate handler for
2 the ContactTextBoxSelectionChanged event.

- 3 • ContactTextBoxTextChangedEventHandler – A delegate handler for the
4 ContactTextBoxTextChanged event.
- 5 • ContactTextBoxTextResolvedEventHandler – A delegate handler for the
6 TextResolvedToContact event.
- 7 • ContentChangedDelegate – Delegate for the ContentChangedEvent.
- 8 • ContextMenuEventHandler - The callback type for handling a
9 ContextMenuEvent.
- 10 • DragDeltaEventHandler – This delegate is used by handlers of the
11 DragDeltaEvent event.
- 12 • IncludeContactEventHandler – Handler for
13 ContactPickerDialog.IncludeContact event.
- 14 • ItemsChangedEventHandler – The delegate to use for handlers that receive
15 ItemsChangedEventArgs.
- 16 • OpenedEventHandler – Handler for ContactPickerDialog.Opened event.
- 17 • PaginationCompleteDelegate – Delegate for the PaginationCompleteEvent.
- 18 • PaginationProgressDelegate – Delegate for the PaginationProgressEvent.
- 19 • ScrollChangeEventHandler – This delegate is used by handlers of the
20 ScrollChangeEvent event.
- 21 • ScrollChangeEventHandler – This delegate is used by handlers of the
22 ScrollChangeEvent event.
- 23 • ScrollChangeEventHandler – This delegate is used by handlers of the
24 ScrollChangeEvent event.
- 25 • ScrollChangeEventHandler – This delegate is used by handlers of the
26 ScrollChangeEvent event.

- 1 • SelectionChangedEventHandler – The delegate type for handling a
2 selection changed event.
- 3 • TextChangedEventHandler – The delegate to use for handlers that receive
4 TextChangedEventArgs.
- 5 • ToolTipEventHandler – The callback type for handling a ToolTipEvent.
- 6 • ValueChangedEventHandler – This delegate is used by handlers of the
7 ValueChangedEvent event.
- 8 • VisibleChangedEventHandler – This delegate is used by handlers of the
9 VisibleChangedEvent event.
- 10 • VisibleChangedEventHandler – This delegate is used by handlers of the
11 VisibleChangedEvent event.

12
13 Another namespace, System.Windows.Controls.Atoms, is a sub-namespace
14 of the System.Windows.Controls namespace. System.Windows.Controls.Atoms
15 includes associated controls, event arguments and event handlers. The following
16 list contains example classes associated with the System.Windows.Controls.Atoms
17 namespace.

- 18 • PageBar – Represents a scrollable pagination control.
- 19 • PageElement – Renders a specific page of paginated content. The page to
20 be rendered is specified by the PageSource property.
- 21 • PageHoveredEventArgs – PageHoveredEventArgs provides information
22 about where the mouse pointer is hovering.
- 23
- 24
- 25

1 PageScrolledEventArgs – The PageScrolledEventArgs contains info
2 pertaining to the PageScrolled Event.

- 3 • PageSelectedEventArgs – The PageSelectedEvent is fired when a new
4 row/column range selection is made.
- 5 • PageSelector – PageSelector: Allows the user to select a range of
6 rows/columns of pages to be displayed.
- 7 • PageSource – Identifies the source of the content to be paginated. It also
8 provides properties and methods for formatting paginated content.
9

10
11
12 The following list contains example delegates associated with the
13 System.Windows.Controls.Atoms namespace.

- 14 • PageHoveredEventHandler – This delegate is used by handlers of the
15 PageHoveredEvent event.
- 16 • PageScrolledEventHandler – This delegate is used by handlers of the
17 PageHovered event.
- 18 • PageSelectedEventHandler – This delegate is used by handlers of the
19 PageSelectedEvent event.
20

21
22
23 A System.Windows.Controls.Primitives namespace is another sub-
24 namespace of the System.Windows.Controls namespace. As mentioned above,
25 the Primitives sub-namespace includes controls that are intended to be used as

1 primitives by other more complex controls. The following list contains example
2 classes associated with the System.Windows.Controls.Primitives namespace.

- 3 • ButtonBase – When overridden in a derived class, defines the relevant
4 events and properties, and provides handlers for the relevant input events.
- 5 • Popup – A control that creates a fly-out window that contains content.
- 6 • RangeBase – Represents the base class for elements that have a specific
7 range. Examples of such elements are scroll bars and progress bars. This
8 class defines the relevant events and properties, and provides handlers for
9 the events.
- 10 • RepeatButton – RepeatButton control adds repeating semantics of when the
11 Click event occurs.
- 12 • ScrollArea – ScrollArea is the effective element for scrolling. It contains
13 content that it clips and provides properties to expose the content's offset
14 and extent. It also provides default input handling such that scrolling can
15 be driven programatically or via keyboard or mouse wheel.
- 16 • ScrollBar – The ScrollBar class.
- 17 • Selector – The base class for controls that select items from among their
18 children.
- 19 • Slider – The Slider class.
- 20 • Thumb – The thumb control enables basic drag-movement functionality for
21 scrollbars and window resizing widgets.
- 22 • Thumb – The thumb control enables basic drag-movement functionality for
23 scrollbars and window resizing widgets.
- 24 • Thumb – The thumb control enables basic drag-movement functionality for
25 scrollbars and window resizing widgets.

1
2 “IEnsureVisible” is an example interface associated with the
3 System.Windows.Controls.Primitives namespace. IEnsureVisible is implemented
4 on a visual to scroll/move a child visual into view.
5

6 The following list contains example enumerations associated with the
7 System.Windows.Controls.Primitives namespace.

- 8 • ArrowButtonStates –
- 9
- 10 • CloseModeType – Describes how a popup should behave to various mouse
11 events.
- 12 • Part – The Part enumeration is used to indicate the semantic use of the
13 controls that make up the scroll bar.
- 14 • PartStates – ScrollBar Part States.
- 15
- 16 • PlacementType – Describes where a popup should be placed on screen.
- 17 • SizeBoxStates –
- 18
- 19

20 Documents Namespace

21 A documents namespace 312 is a collection of semantic and formatting
22 elements that are used to create richly formatted and semantically rich documents.
23 In one embodiment, an “element” is a class that is primarily used in conjunction
24 with a hierarchy of elements (referred to as a “tree”). These elements can be
25

1 interactive (e.g., receiving user input via keyboard, mouse or other input device),
2 can render images or objects, and can assist with the arrangement of other
3 elements. Example elements include a “Block” element that implements a generic
4 block, a “Body” element that represents content that includes the body of a table, a
5 “Cell” element that contains tabular data within a table, a “Header” element that
6 represents the content included in the header of a table, and a “PageBreak”
7 element that is used to break content across multiple pages.

8 The following list contains example classes exposed by the
9 System.Windows.Documents namespace.

- 10 • AdaptiveMetricsContext - AdaptiveMetricsContext provides the root
11 element for adaptive-flow-format documents. Once a child panel is
12 encapsulated in an AdaptiveMetricsContext element, the content of the
13 panel is processed by the Reading Metrics Engine (RME). The size of the
14 child panel is used to calculate the number and size of any columns as well
15 as optimum font sizes and line heights.
- 16 • Block - Implements a generic block element that does not induce default
17 rendering behavior.
- 18 • BlockElement - Implements a base class for all Block elements.
- 19 • Body - Represents the content that comprises the body of a Table element.
- 20 • Bold - Implements a Bold element derived from Inline.
- 21
- 22
- 23
- 24
- 25

1 BreakRecord - Stores information necessary to continue formatting
2 paginated content across page breaks. Inherit from this class to provide
3 pagination support. This is an abstract class.

- 4 • Cell – Cells contain tabular data within a Table. Cell elements are
5 contained within a Row.
- 6
- 7 • CellCollection - Ordered collection of table cells.
- 8
- 9 • Column – The Column element is used to apportion the contents of a
10 GridPanel or Table.
- 11
- 12 • ColumnCollection - A ColumnCollection is an ordered collection of
13 Columns.
- 14
- 15 • ColumnResult - Represents a column's view-related information.
- 16
- 17 • ContainerParagraphResult – Provides access to calculated layout
18 parameters for a Paragraph object which contains only other Paragraph
19 objects.
- 20
- 21 • ContentPosition - Represents the position of content within a paragraph.
22 Inherit from this class to describe the position of associated content. This is
23 an abstract class.
- 24
- 25 • Document - The purpose of the Document class is to decouple the content
of a document from the UI "chrome" that surrounds it. "Decoupling"
means that you can author a document without thinking about (and without
committing to) its UI. The Document class holds document content,
typically a TextPanel or a FixedPanel and its children. A visual tree (by

1 default, a PageViewer) is associated with this element through the WPP
2 control styling mechanism.

- 3 • DocumentPage - Represents layout information for a control associated
4 with a page of a document subject to pagination. Inherit from this class to
5 implement to describe the layout information for these controls. This is an
6 abstract class.
- 7
- 8 • DocumentPageParagraphResult - Provides access to calculated layout
9 parameters for objects affected by pagination.
- 10 • FindEngine – Base class for find algorithms.
- 11
- 12 • FindEngineFactory – Find algorithms factory.
- 13
- 14 • FixedPage - Provides access to a single page of content within a fixed-
15 format layout document.
- 16 • Footer – Represents the content that comprises the footer of a Table
17 element.
- 18 • Header - Represents the content that comprises the header of a Table
19 element.
- 20 • Heading – Implements a block-level element that renders text as a heading.
- 21
- 22 • HyphenationDictionary - HyphenationDictionary represents a dictionary for
23 the purpose of providing hyphenation support within applications. It can
24 contain both an inline dictionary and a reference to an external dictionary.
25

1 The inline dictionary has higher priority and will be applied before entries
2 in the external dictionary.

- 3 • Hyphenator – The Hyphenator object maintains reference to hyphenation
4 data within a HyphenationDictionary and also performs hyphenation.
- 5 • Inline - Implements a generic Inline element that does not induce any
6 default rendering behavior.
- 7 • InlineElement - Implements a generic inline element as base class for all
8 inline elements.
- 9 • Italic – Implements an Italic element derived from Inline.
- 10 • LineBreak - Represents a markup element that forces a line break.
- 11 • LineResult - Provides access to calculated information of a line of text.
- 12 • List - Implements a List element. Lists are block-level elements designed
13 to be formatted with markers such as bullets or numbering.
- 14 • ListElementItem - Implements a ListElementItem, which supports markers
15 such as bullets or numbering.
- 16 • Note - Implements a Note element, which is analagous to the note element
17 in HTML.
- 18 • PageBreak - Represents a markup element used to break content across
19 various pages.
- 20
- 21
- 22
- 23
- 24
- 25

1 PageDescriptor - Implements PageDescriptor, which stores information
2 necessary to create paginated layout.

- 3 • Paragraph - Implements a block-level element used to render text in a
4 paragraph. Rendering behavior is analagous to that of the paragraph
5 element in HTML.
- 6 • ParagraphResult - Provides access to calculated layout parameters for a
7 Paragraph object.
- 8 • Row – Defines a row within a GridPanel or Table element.
- 9 • RowCollection – RowCollection represents an ordered collection of Rows.
- 10 • RowGroup – Specifies property defaults for a group of rows in a Table or
11 GridPanel.
- 12 • Section - Implements a generic container element. Rendering behavior is
13 analagous to the div element in HTML.
- 14 • SmallCaps - Implements an inline SmallCaps element. SmallCaps are
15 typographic forms that render as small capital versions of letters for
16 emphasis, as in a title.
- 17 • Subscript - Represents an inline Subscript element. Subscript characters
18 are written immediately below, below and to the left, or below and to the
19 right of other characters.
- 20 • Subscript - Represents an inline Subscript element. Subscript characters
21 are written immediately below, below and to the left, or below and to the
22 right of other characters.

1 Superscript - Represents an inline Superscript element. Superscript
2 characters are typically letters or numbers and render immediately above,
3 above and to the left, or above and to the right of other characters.

- 4 • Table – Table is used to display complex data in tabular form using a
5 markup language (e.g., "XAML").
- 6
- 7 • TextArray - Base API for text access and manipulation.
- 8
- 9 • TextChangedEventArgs - The TextChangedEventArgs defines the event
10 arguments sent when a TextArray is changed.
- 11
- 12 • TextElement - TextElement provides TextRange facilities for the TextTree.
13 It is an immutable, continuous TextRange with fixed endpoints. It provides
14 ContentElement Input, Focus and Eventing support. It also provides
15 DependencyObject property support.
- 16
- 17 • TextNavigator - This can enumerate text content. Implements a movable
18 TextPosition. It can move by text run or be positioned at a know location
19 in text.
- 20
- 21 • TextParagraphResult - Provides access to calculated layout parameters for
22 text, including floated objects and figures.
- 23
- 24 • TextPosition - This is an object representing a certain position in a
25 TextArray. A compact object representing a position in text automatically
maintains position when text changes. Comparison operations are only
applicable to positions within same TextArray (same Context) TextPosition
can be static or movable. IsChangeable property tells the kind of position.

1 TextRange - TextRange is an abstract class providing generic association of
2 zero or more subranges with properties. Subrange manipulation is defined
3 on derived classes.

- 4 • TextRangeMovable – TextRangeMovable is an abstract class for movable
5 TextRanges. It adds the ability to move the start and end points based on
6 TextUnits.
- 7 • TextTreeChangedEventArgs - The TextChangedEventArgs defines the
8 event arguments sent when a TextArray is changed.
- 9 • TextTreeDumper - TreeDumper is a tree test class that is public due to
10 packaging issues.
- 11 • TextTreeNavigator - This is an object representing a certain moveable
12 position in a TextTree. It is a specific implementation of TextNavigator for
13 use only in the TextTree.
- 14 • TextTreePosition - This is an object representing a certain immutable
15 position in a TextTree. It is a specific implementation of TextPosition for
16 use only in the TextTree.
- 17 • TextTreeRange - Provides TextRange facilities for the TextTree. It is a
18 mutable, continuous TextRange with movable endpoints.
- 19 • TextTreeRangeContentEnumerator - Enumerator on object children directly
20 under a TextTreeRange.
- 21 • TextTreeRangeContentEnumerator - Enumerator on object children directly
22 under a TextTreeRange.
- 23 • TextTreeRangeContentEnumerator - Enumerator on object children directly
24 under a TextTreeRange.

25 TextUnit - Extensible unit of text navigation.

1 TextUnits - Commonly used text units for TextPosition and TextRange.

2 Typography - Provides access to a rich set of OpenType typography
3 properties.

- 4 • UIElementParagraphResult - The ParagraphResult for a paragraph which is
5 composed entirely of a UIElement. Used for Floaters, Figures and
6 embedded block level UIElements.
- 7 • Underline – Implements an Underline element derived from InlineElement.

8
9
10
11 The following list contains example interfaces associated with the
12 System.Windows.Documents namespace.

- 13 • IDocumentContentHost - Implement this interface on a content host so that
14 children of that host can notify the host when content is changing.
- 15 • IDocumentFormatter - Implement this interface on an element to provide
16 support for document features such as pagination.
- 17 • ITextDocumentResult - Implement this interface to maintain column
18 information for a document.
- 19 • ITextParagraphResult - Implement this interface to provide text and
20 positioning information for text paragraphs.

1 The following list contains example enumerations associated with the
2 `System.Windows.Documents` namespace.

- 3 • `ElementEdge` - This identifies the edge of an object where a `TextPosition` is
4 located.
- 5 • `FindAdvancedOptions` - The advanced search options used by
6 `FindAlgorithm` (search initialization) and
7 `TextRangeMovable/TextSelection` (simplified search execution) classes.
- 8 • `FindOptions` - The simplified search options used by `TextBox.Find`
9 methods.
- 10 • `LogicalDirection` - `LogicalDirection` defines a logical direction for
11 movement in text. It is also used to determine where a `TextPosition` will
12 move when content is inserted at the `TextPosition`.
- 13 • `TextArrayRunType` - This identifies the run where a `TextPosition` is
14 located, taking `LogicalDirection` into account.
- 15 • `TextChangeOptions` - Possible text changes for `CanChangeText`.
- 16 • `TextMoveOptions` - This controls the movement of `TextNavigator` by
17 specifying conditions to halt navigation.
- 18 • `TextMoveOptions` - This controls the movement of `TextNavigator` by
19 specifying conditions to halt navigation.
- 20 • `TextMoveOptions` - This controls the movement of `TextNavigator` by
21 specifying conditions to halt navigation.

22 The following list contains example delegates associated with the
23 `System.Windows.Documents` namespace.

ObjectCloneDelegate - Callback method to provide a clone or copy of a DependencyObject when a portion of a TextArray is being copied or moved.

- TextChangedEventHandler - The TextChangedEventHandler delegate is called with TextChangedEventArgs every time content is added to or removed from the TextTree.

Shapes Namespace

A shapes namespace 314 is a collection of vector graphics elements that is used to create images and objects. The use of vector graphics elements allows the elements to be easily resized to fit the requirements of a particular interface or display device. Example elements include an “Ellipse” element that draws an ellipse, a “Line” element that draws a straight line between two points, a “Rectangle” element that draws a rectangle, and a “Polygon” element that draws a polygon as a connected series of lines that form a closed shape.

The following list contains example classes exposed by the System.Windows.Shapes namespace.

- Ellipse – Draws an ellipse.
- Glyphs - Represents a glyph shape in a markup language such as "XAML".
Glyphs are used to represent fonts.
- Line – Draws a straight line between two points.
- Path – Draws a series of connected lines and curves.

1 Polygon – Draws a polygon (a connected series of lines that forms a closed
2 shape).

- 3 • Polyline – Draws a series of connected straight lines.
- 4 • Rectangle – Draws a rectangle.
- 5 • Shape – An abstract class that provides base functionality for shape
6 elements, such as ellipse, polygon and rectangle.
7

8 9 10 Data Namespace

11 A data namespace 316 includes classes and interfaces used to bind
12 properties of elements to data sources, data source classes, and data-specific
13 implementations of data collections and views. These classes and interfaces are
14 also used to handle exceptions in data entry and allow runtime creation of a user
15 interface based on information in various data sources. Data can be displayed in
16 textual form or can be utilized to change the formatting of the display, such as
17 displaying dollar amounts in red if they are negative. Example classes include a
18 “Bind” class that represents a binding declaration object that manages bindings
19 between a dynamic property user interface and source data, and an
20 “XmlDataSource” class that serves as a data source for data binding to XML
21 content nodes.

22 Object-oriented applications typically represent data by classes that define
23 both the value of a piece of data and the operations that can be performed on that
24 data. The term “data item” refers to one such object. Applications can handle
25

1 individual data items or collections of data items. They may use data items in
2 three ways: (a) converting data from external sources such as file systems, remote
3 servers, databases, etc. into the corresponding in-memory data items, and
4 converting modified data items back into the form expected by these sources; (b)
5 operating on the data items using a combination of data-centric and application-
6 centric logic; (c) presenting the data embodied by the data items to the user
7 through a user interface. Data namespace 316 provides support for the first and
8 third of these tasks.

9 The first task, obtaining data from external sources, is supported by “data
10 source” objects. A data source object is typically defined as a page-wide or
11 application-wide resource, and serves as the gateway to the data. Data sources
12 implement an `IDataSource` interface, which defines a standard mechanism by
13 which the classes in the data namespace get access to the data. A particular data
14 source object implements logic for retrieving the actual data by using mechanisms
15 appropriate for the particular source. In one embodiment, the data namespace
16 includes four data source classes:

- 17 1. `XmlDataSource`, for retrieving data represented as XML
- 18 2. `SqlDataSource`, for retrieving data from SQL databases, such as
19 Microsoft SQL Server
- 20 3. `WinFSDataSource`, for retrieving data from the WinFS service
- 21 4. `ObjectDataSource`, for retrieving data from an arbitrary object
22 defined by the application

23 Applications can also define their own data source classes that are tailored to
24 special-purpose sources.
25

1 A data source class is responsible for retrieving data from an external
2 source and converting it into one or more data items suitable for use by the
3 binding classes. If a collection of data items is needed, the application can use any
4 of the standard collection classes from the .Net Framework such as Array,
5 ArrayList, Hashtable, etc., any of the data-centric collection classes from the
6 System.Data namespace such as Dataset, or a data-centric collection class from the
7 data namespace such as ArrayListDataCollection. The latter classes support
8 change notifications; i.e., when the application changes the collection by adding an
9 item, removing an item, sorting the collection, etc., the collection sends a
10 notification. The binding classes listen for these notifications and automatically
11 update the user interface to reflect the change.

12 Once the data has been converted to in-memory data items, the application
13 can perform computations using the items and can modify the items as a result of
14 the computations. These actions are performed using a combination of data-
15 centric operations (defined by the data item classes) and application-centric
16 operations (defined by the application itself). The actions may be initiated by the
17 application automatically, or in response to an action of the user. Special support
18 or cooperation from the data namespace is not necessary, thereby providing a clean
19 separation of logic and presentation within the application.

20 The third data-related task, presenting the data through the user interface, is
21 supported by the “binding” classes of the data namespace. These classes enable
22 the application to describe the correspondence (binding) between a data item
23 property (the source) and a user interface property (the target). The term data-
24 binding (or simply binding) refers to the establishment of such a correspondence.
25 For example, an application may choose to data-bind the Text property of a

1 Textbox control to the CustomerName property of a data item. Having done so,
2 the control will automatically display the customer's name, updating the display
3 whenever the application changes the data item, and updating the data item
4 whenever the user types a new name into the control.

5 This kind of correspondence is described using the Bind class, and
6 implemented using the Binding class. Any number of UI properties can share the
7 same description (Bind), but each property has its own unique Binding that holds
8 the state for that particular instance. The description includes the following
9 information about the desired correspondence:

- 10 • Path – the name of the data item property to use as the source of the
11 binding. This can be a simple property name, or a more complicated
12 expression involving sub-objects and indexers (e.g., when the source
13 property has a value of complex type) such as
14 “ShippingAddress.Line[2]”. When the data source is XML, the path
15 is an XPath expression.
 - 16 • BindType – whether the correspondence is one-way, two-way, or
17 one-time. In a one-way binding, changes to the data item cause
18 updates to the user interface property; data flows one way – from
19 the source to the target. In a two-way binding the data flows in both
20 directions; in addition to the one-way behavior, changes to the user
21 interface property cause updates to the data item property. In a one-
22 time binding, the data item property is used to initialize the user
23 interface property, but changes do not propagate in either direction.
- 24 Source – a description of where to obtain the source data item. This
25 can be from a data source object, from some other user interface

1 element, or from the value of the target element's DataContext
2 property.

3 UpdateType – when to update the source property in a two-way
4 binding: one of Immediate, OnLostFocus, or Explicit. Immediate
5 updates happen as soon as the user interface property changes.
6 OnLostFocus updates are delayed until the target element loses
7 keyboard focus – this is appropriate for a TextBox control, to avoid
8 the expense of updating after every keystroke. Explicit updates
9 happen when the application explicitly calls for them.

- 10 • Transformer – an object that implements the IDataTransformer
11 interface. This gives the application a way to modify the data item
12 value before using it in the user interface. The modification can be a
13 simple type conversion (e.g., in a binding of the Background
14 property to the BalanceOwed data property, the application can
15 convert a negative balance to a red background and a positive
16 balance to green), or an application-specific conversion (e.g., in a
17 binding of the Text property to the NetWorth data property, the
18 application can display “Rich” if the NetWorth exceeds \$1M,
19 “Bourgeois” if the NetWorth lies between \$100K and \$1M, and
20 “Poor” if the NetWorth is less than \$100K). Transformers are a
21 simple yet powerful tool that help separate presentation from data.

22 All bindings except one-time rely on getting notified when the data
23 property changes, so that the corresponding change can be made to the user
24 interface. The binding classes recognize the IPropertyChange interface (from the
25

System.ComponentModel namespace) as one way of implementing the required notifications.

The following tables list the members exposed by the System.Windows.Data namespace.

Classes

ArrayListCollectionView	Encapsulates the collection view support for the ArrayListDataCollection collection class. This class cannot be inherited.
ArrayListDataCollection	Provides a built-in implementation of an array-list data collection with an underlying collection-view interface. It also implements ICollectionChange to provide notification when items are added, items are removed, or the whole collection is refreshed.
Bind	Represents a bind declaration object, used to manage bindings between a dynamic property user interface (UI) and source data.
Binding	Provides access to the single run-time instance of a binding. This class cannot be inherited.
BindingListCollectionView	A collection view class used for Microsoft® ActiveX® Data Objects (ADO) data views.
CollectionContainer	Objects of this class hold an existing collection structure—for example, an ArrayListDataCollection or some other DataSet inside the ItemCollection.
ContextAffinityCollectionView	Implements a collection view that includes checks for context affinity.
DataContextObjectRef	Supports object references to objects being used as data context for a binding. This class cannot be inherited.
DataSourceObjectRef	Supports object references to data

1		sources. This class cannot be inherited.
2		Encapsulates arguments for data
3	DataTransferEventArgs	transfer events. The events are routed
4		events that are handled specifically by a
5		designated handler based on the
6		DataTransferEventHandler delegate.
7		Represents an object reference to an
8	ElementObjectRef	element, with the object being specified by
9		its element ID. This class cannot be
10		inherited.
11		Represents an explicit object
12	ExplicitObjectRef	reference to an element. This class cannot
13		be inherited.
14		Implements a collection view for
15	ListCollectionView	collections based on ICollection .
16		Serves as a data source for data
17	ObjectDataSource	binding. Bindable data items can be
18		specified as common language runtime
19		types.
20		The abstract class that is used as the
21	ObjectRef	parent class of ElementObjectRef ,
22		ExplicitObjectRef , and TypeObjectRef .
23		Objects of this class hold the
24	ParameterCollection	collection of named parameters (with their
25		corresponding values) for an
26		SqlDataSource .
27		This class represents a single select
28	QueryCommand	statement to be submitted to the database.
29		Encapsulates the arguments passed
30		either in the RefreshCompleted event of
31	RefreshCompletedEventArgs	ObjectDataSource , or in the
32		RefreshCompleted event of
33		XmlDataSource .
34		A list of sql commands and the
35	SqlCommandList	names of the tables that they should be
36		used to fill.
37		SqlDataSource gets data from a
38	SqlDataSource	Microsoft SQL Server for use in
39		databinding.
40		Allows resource reference to a
41	TransformerSource	transformer class that is defined as code-

1		behind in the current application.
2	TypeObjectRef	Supports object reference by type. This class cannot be inherited.
3	WinFSDataSource	The WinFSDataSource facilitates databinding of data stored in WinFS with Avalon applications
4		XmlDataNamespaceManager Class
5	XmlDataNamespaceManager	Used to declare namespaces to be used in Xml data binding XPath queries
6		Serves as a data source for data
7	XmlDataSource	binding to Extensible Markup Language (XML) content nodes.
8		Declares an individual namespace
9	XmlNamespace	within an XML data source.
10		
11	Interfaces	
12		Used to create classes that declare filtering criteria
13	IContains	for collection views.
14		Supports creation of data source objects. Data
15	IDataSource	source objects are used for common representation of data for data binding.
16		Provides methods that enable client-side
17	IDataTransformer	transformation of bound data.
18		
19	Enumerations	
20		Describes special properties of a binding. See
21	BindFlags	Using Bind Declarations for "Longhorn" markup language (code-named "XAML") usage. See
22		BindType for the enumeration that is used to specify binding type (one-way, two-way and so on).
23	BindStatus	Status of a Binding.
24		Describes how changes in data values will
25	BindType	transfer to and from source properties and target properties of a binding.
	SqlDataSourceMode	The enumeration of the possible modes that

1 SqlDataSource can have. The mode determines what
2 sort of data is returned when the application retrieves
the value from the Data property.

3 UpdateType Specifies when updates to the data source
4 (target-to-source data transfer) should occur in a
binding. Setting these values will only be relevant if a
5 binding's BindType is set to TwoWay (or left as the
default).

6 Delegates

7 DataChangedEventHandler Represents the method that handles
8 the DataChanged event raised by data
sources that implement IDataSource.

9 DataTransferEventHandler Represents the method that handles
a data transfer event raised by Binding.

10 RefreshCompletedEventHandler Represents the method that handles
11 the ObjectDataSource.RefreshCompleted
and XmlDataSource.RefreshCompleted
12 events.

13 Media Namespace

14 A media namespace 318 provides various media classes. Application
15 developers as well as component developers may use these classes to develop
16 various presentation functionality. Example classes in media namespace 318
17 include an "ImageEffect" class that permits certain imaging effects (e.g., blur and
18 grayscale), and a "Brush" class that provides a mechanism for filling an area using
19 solid colors, gradients, images, video, and the like.

20 The media namespace 318 includes a sub-namespace
21 System.Windows.Media.Animation that includes services that allow a developer
22 to animate properties and coordinate a set of animations with a set of timelines.
23 An animation is an object that changes a value over a period of time. Animation
24 effects include moving an object on the display, and changing the size, shape, or
25

1 color of an object. Multiple animation classes are provided to implement various
2 animation effects. Effects can be achieved by associating an animation with an
3 element's property value. For example, to create a rectangle that fades in and out
4 of view, one or more animations are associated with the opacity property of the
5 rectangle.

6 The media namespace 318 also includes a sub-namespace
7 System.Windows.Media.TextFormatting that provides various text services. For
8 example, a "TextFormatter" text engine provides services for breaking text lines
9 and formatting text presented on a display. "TextFormatter" is capable of handling
10 different text character formats and paragraph styles as well as handling
11 international text layout.

12 The following tables list example members exposed by the
13 System.Windows.Media namespace.

14 Classes

15 ArcSegment

Represents an elliptical arc between
16 two points.

17 AudioData

Enables playing of audio files
according to the state of a time node.

18 AudioDataConverter

AudioDataConverter

19 BezierSegment

Represents a cubic Bézier curve
drawn between two points.

20 Brush

Provides a generic means for filling an
area using solid colors (SolidColorBrush),
21 gradients (LinearGradientBrush,
22 RadialGradientBrush), images (ImageBrush),
video, and more.

23 BrushConverter

Used to convert a Brush object to or
from another object type.

24 Brushes

Implements a set of predefined solid
25 colors.

1	CloseSegment	Represents a line that connects the last point of a PathFigure object with its starting point.
2		
3	CodecFilter	Filter for enumerating codecs. Only those codecs that match the properties will be enumerated.
4		
5	CodecInfo	Information about a specific codec and a factory for creating the codec. This is returned from the codec enumerator.
6	ColorCollection	
7	ColorCollectionConverter	ColorCollectionConverter - Converter class for converting instances of other types to and from ColorCollection instances.
8		
9	ColorContext	
10	ColorConverter	Used to convert a Color object to or from another object type.
11	Colors	Implements a set of predefined colors.
12	ContainerVisual	Manages a collection of Visual objects.
13	DashArrays	DashArrays - The DashArrays class is static, and contains properties for well known dash styles.
14		
15	DoubleCollection	
16	DoubleCollectionConverter	DoubleCollectionConverter - Converter class for converting instances of other types to and from DoubleCollection instances.
17		
18	Drawing	A Drawing is a list of 2d drawing primitives.
19	DrawingBrush	DrawingBrush - This TileBrush defines its content as a Drawing
20	DrawingContext	Drawing context.
21	DrawingVisual	Visual that contains graphical content to be drawn.
22	EllipseGeometry	Represents the geometry of a circle or ellipse.
23	FontFamily	Font family
24	FormattedText	The FormattedText class is a part of Avalon MIL easy text API, which is targeted at programmers needing to add some simple
25		

1		text to a MIL visual.
2		An abstract class that provides base
3	Geometry	functionality for all geometry classes, such as
4		EllipseGeometry, RectangleGeometry, and
5		PathGeometry. The Geometry class of
6		objects can be used for clipping, hit-testing,
7		and rendering 2-D graphic data.
8		Represents a collection of Geometry
9	GeometryCollection	objects.
10	GetPageEventArgs	class GetPageEventArgs
11	GlyphRun	Glyph run class
12		Physical font face corresponds to a
13	GlyphTypeface	font file on the disk
14		An abstract class that describes a
15	GradientBrush	gradient fill. Classes that derive from
16		GradientBrush describe different ways of
17		interpreting gradient stops.
18	GradientStop	Describes the location and color of a
19		transition point in a gradient.
20	GradientStopCollection	Represents a collection of
21		GradientStop gradient stops.
22	HitTestParameters	This is the base class for packing
23		together parameters for a hit test pass.
24	HitTestResult	This base returns the visual that was
25		hit during a hit test pass.
26	HwndInterop	HwndInterop
27	HwndVisual	
28	HyphenationCandidate	Describes one Hyphenation candidate.
29	ICCProfile	
30		Fills an area with an image. This class
31	ImageBrush	may be used to specify images as the fill or
32		background of other objects.
33	ImageCodecCollection	The collection of codecs (actually
34		CodecInfos) on the system.
35	ImageCodecEnumerator	The enumerator for Image frames.
36	ImageColorTransform	ImageColorTransform Performs color
37		management on an imaging pipeline.
38	ImageData	Contains an image and related data.
39	ImageDataBuilder	This object is used to build an

1		ImageData object.
2		ImageDecoder is a container for
3	ImageDecoder	image frames. Each image frame is an
4		ImageSource. Unlike ImageSource,
5		ImageDecoder is NOT an immutable object
6	ImageDecoderBmp	and can be re-initialized to a different image
7	ImageDecoderGif	stream. However, any ImageSources (frames)
8	ImageDecoderIcon	that it returns should be immutable.
9	ImageDecoderInternal	The built-in Microsoft Bmp (Bitmap)
10	ImageDecoderJpeg	Decoder.
11	ImageDecoderPng	The built-in Microsoft GIF Decoder.
12	ImageDecoderTiff	The built-in Microsoft Icon Decoder.
13	ImageEffect	For internal use only.
14		The built-in Microsoft Jpeg Decoder.
15		The built-in Microsoft PNG Decoder.
16		The built-in Microsoft Tiff Decoder.
17		The ImageEffect class is the base class
18	ImageEffectBlur	for all imaging effects (blur, grayscale, etc)
19		It's possible for an effect to not have any
20		inputs but an effect should have at least one
21		output. The default implementations of things
22		assume this. If a derived effect is going to
23	ImageEffectFlipRotate	play with Output/Outputs be sure that at least
24	ImageEffectGammaCorrect	one is there.
25	ImageEffectGlow	Gaussian blur effect. It is a single
	ImageEffectGrayscale	input, single output effect. Warning: If the
		effect is being scaled (i.e. Input.ScaleX or
		Input.ScaleY isn't 1) and Expand is true, then
		it's possible for the output dimensions to be
		larger or smaller than PixelWidth and
		PixelHeight. Adjust the pixel buffer fed to
		copy to avoid problems.
		This effect can flip an image in X or Y
		and rotate by multiples of 90 deg
		This effect changes the gamma of an
		image
		Performs a glow effect. It is a single
		input, single output effect.
		Converts an image to grayscale. It is a
		single input, single output effect.

1	ImageEffectNegate	Negates an image. It is a single input, single output effect.
2	ImageEffectSharpen	Unsharp mask. It is a single input, single output effect.
3	ImageEffectSource	ImageEffectSource class
4	ImageEffectSourceCollection	implementation
5	ImageEffectTint	The collection of image effect outputs
6		Tint constructor. It is a single input, single output effect.
7		ImageEncoder collects a set of frames (ImageSource's) with their associated thumbnails and metadata and saves them to a specified stream. In addition to frame-specific thumbnails and metadata, there can also be an image-wide (global) thumbnail and metadata, if the codec supports it.
8	ImageEncoder	
9		
10	ImageEncoderBmp	Built-in Encoder for Bmp files.
11	ImageEncoderGif	Built-in Encoder for Gif files.
12		ImageEncoderInternal collects a set of frames (ImageSource's) with their associated thumbnails and metadata and saves them to a specified stream. In addition to frame-specific thumbnails and metadata, there can also be an image-wide (global) thumbnail and metadata, if the codec supports it.
13	ImageEncoderInternal	
14		
15		
16	ImageEncoderJpeg	Built-in Encoder for Jpeg files.
17	ImageEncoderPng	Built-in Encoder for Png files.
18	ImageEncoderTiff	Built-in Encoder for Tiff files.
19		ImageExchangeMetaData This class is used to access and set metadata for ImageFiles which have Exif style metadata. MetaData is stored as Key/Value pairs, where Keys are not necessarily unique. This class provides generic access to all meta data within an image, as well as exposes CLR properties for certain well-known properties.
20	ImageExchangeMetaData	
21		
22		
23	ImageExchangeProperty	ImageExchangeProperty - a tuple of an ImageExchangeID and the object which is the value of that property
24		
25	ImageMetaData	ImageMetaData This class is used to

1		access and set metadata for Images. This class also exposes a CodecMetaData property which exposes a codec-specific means of accessing the metadata for this image.
2		
3	ImagePalette	ImagePalette class
4	ImageSizeOptions	Sizing options for an image. The resulting image will be scaled based on these options.
5		
6	ImageSource	Defines the methods, properties, and events for the imaging pipeline, including decoders and effects.
7		
8	ImageSourceCollection	The collection of codecs (actually ImageSource's) on the system.
9	ImageSourceConverter	ImageSourceConverter
10	IntegerCollection	
11	IntegerCollectionConverter	IntegerCollectionConverter - Converter class for converting instances of other types to and from IntegerCollection instances.
12		
13	LinearGradientBrush	Defines a linear gradient used to fill an area.
14	LineGeometry	Represents the geometry of a line.
15	LineSegment	Represents a line between two points. Unlike LineGeometry objects, LineSegment should be contained within a PathFigure.
16		
17	MatrixTransform	Creates an arbitrary affine matrix transformation used to manipulate objects or coordinate systems in a two-dimensional plane.
18		
19	MediaData	MediaData. Use to playback Audio/Video content.
20	MediaSystem	The MediaSystem class controls the media layer.
21		
22	NineGridBrush	Fills an entire area with an image. Portions of the image are stretched to fit within defined margins.
23		
24	PathFigure	Represents a sub-section of a geometry, a single connected series of two-dimensional geometric segments.
25	PathFigureCollection	

1	PathFigureConverter	PathFigureConverter
2	PathGeometry	Represents a complex shape that may be composed of arcs, curves, ellipses, lines, and rectangles.
3	PathGeometryConverter	PathGeometryConverter
4		An abstract class that represents a segment of a PathFigure object. Classes that derive from PathSegment, such as
5	PathSegment	ArcSegment, BezierSegment, and
6		LineSegment, represent specific types of geometric segments.
7		
8	PathSegmentCollection	Represents a list of PathSegment objects.
9	PathSegmentConverter	PathSegmentConverter
10	Pen	Describes how a shape is outlined.
11	PixelFormats	PixelFormats - The collection of supported Pixel Formats
12	PointCollection	
13	PointCollectionConverter	PointCollectionConverter - Converter class for converting instances of other types to and from PointCollection instances.
14	PointHitTestParameters	This is the class for specifying parameters hit testing with a point.
15	PointHitTestResult	This class returns the point and visual hit during a hit test pass.
16	PolyBezierSegment	PolyBezierSegment
17	PolyLineSegment	PolyLineSegment
18	PolyQuadraticBezierSegment	PolyQuadraticBezierSegment
19	PrintContext	PrintContext holds state and context for a printer interaction
20	QuadraticBezierSegment	QuadraticBezierSegment
21	RadialGradientBrush	Defines a radial gradient used to fill an object. A focal point defines the beginning of the gradient, and a circle defines the end point of the gradient.
22		
23	RectangleGeometry	Represents the geometry of a rectangle.
24	RetainedVisual	RetainedVisual
25	RotateTransform	Used to rotate an object about a

1		specified point in the two-dimensional x-y plane.
2		Scales an object in the two-dimensional x-y plane, starting from a defined center point. Scale factors are defined in x- and y-directions from this center point.
3	ScaleTransform	
4		
5	SkewTransform	Represents a two-dimensional skew.
6	SolidColorBrush	Represents a solid, uniform fill.
7	StartSegment	StartSegment
8		collection of subline. Subline can be object of one of these types GlyphRun LineOver Inline object
9		Abstract class that describes a way to fill a region with one or more "tiles." Derived classes define the different types of tiles that can be used; for example, the ImageBrush enables you to fill an area with an image.
10	TileBrush	
11		
12		An abstract class that you use as the parent class of all types of transformations in a two-dimensional plane, including rotation (RotateTransform), scale (ScaleTransform), skew (SkewTransform), and translation (TranslateTransform). This class hierarchy differs from the Matrix structure both because it is a class and because it supports animation and enumeration semantics.
13		
14	Transform	
15		
16		
17	TransformCollection	Used to create and manipulate a list of Transform objects.
18		
19	TransformConverter	Used to convert a Transform object to or from another object type.
20	TranslateTransform	Translates an object in the two-dimensional x-y plane.
21	Typeface	A Typeface is a combination of family, weight, style and stretch:
22	VectorCollection	
23		VectorCollectionConverter -
24	VectorCollectionConverter	Converter class for converting instances of other types to and from VectorCollection instances.
25		

1	VideoData	Enables playing of video files according to the state of a time node.
2	VideoDataConverter	VideoDataConverter
3		Base class for all Visual types. It
4	Visual	provides services and properties common to all Visuals, including hit-testing, coordinate transformation, and bounding box calculations.
5		
6	VisualCollection	An ordered collection of Visual objects.
7	VisualManager	Renders a tree of Visual objects to a rendering target, typically a window.
8		
9		
10	Interfaces	
11	IHyphenate	IHyphenate is the interface for Hyphenation Service Provider
12		If this interface is implemented on a class that is
13		derived from a RetainedVisual, the RetainedVisual
14	IRetainedRender	operations in validation mode, i.e. the graphics sub-system will call OnRender in a lazy fashion. (e.g. if the Visual appears for the first time on the screen). Note that OnRender can be called by the system anytime.
15		
16	IVisual	This interface defines the common methods and services available from a Visual object.
17		
18		
19	Enumerations	
20		BrushMappingMode - Enum which
21	BrushMappingMode	describes whether certain values should be considered as absolute local coordinates or whether they should be considered multiples of a bounding box's size.
22		
23	ChannelDescription	Describes order of each channel of pixel data
24	ColorInterpolationMode	ColorInterpolationMode - This determines how the colors in a gradient are interpolated.
25		

1	CombineMode	Specifies the method used to combine two geometric areas.
2	FillRule	
3	GradientSpreadMethod	Specifies how the gradient should be drawn outside of the specified gradient vector or space.
4	HitTestFilterBehavior	Behavior for filtering visuals while hit testing
5	HitTestResultBehavior	Enum controls behavior when a positive hit occurs during hit testing.
6	HorizontalAlignment	The HorizontalAlignment enum is used to describe how content is positioned horizontally within a container.
8	HyphenationRule	Supported Hyphenation Rules.
9	ImagePaletteType	Pre-defined palette types
10	MediaState	Holds the current state of the Media
11	PenDashCap	PenDashCap - Enum which describes the drawing of the ends of a dash within a dashed line.
12	PenLineCap	Describes the shape at the end of a line or segment.
13	PenLineJoin	PenLineJoin - Enum which describes the drawing of the corners on the line.
14	Rotation	The rotation to be applied; only multiples of 90 degrees is supported.
15	StandardColorSpace	
16	Stretch	Stretch - Enum which describes how a source rect should be stretched to fit a destination rect.
18	StyleSimulations	Font style simulation
19	TiffCompressOptions	Compress options for saving TIFF image
20	TileMode	TileMode - Enum which describes the drawing of the ends of a line.
21	VerticalAlignment	The VerticalAlignment enum is used to describe how content is positioned vertically within a container.
22		
23		
24	Structures	
25		

1	CharacterIndexer	This class is a helper to implement named indexers for characters.
2		
3	Color	Represents colors in terms of alpha, red, green, and blue channels.
4		
5	GlyphIndexer	This class is a helper to implement named indexers for glyph metrics.
6		
7	ImageExchangeID	ImageExchangeID - This class is the type which can be used as the key for a property in an ImageMetaData instance. This can be either an integer or a string.
8		
9	ImageExchangeMetaDataEnumer	ImageExchangeMetaDataEnumer
10	ator	ator The enumerator for ImageExchangeMetaData. Contains IEnumerator interface as well as strongly typed versions of the APIs
11		
12	ImageFrameEnumerator	The enumerator for Image frames.
13		
14	ImageMetaDataRational	An ImageMetaDataRational class is represented as a signed numerator and a signed denominator. The effective value of a rational is the numerator / demoninator
15		
16	ImageMetaDataUnsignedRational	A rational class is represented as an unsigned numerator and an unsigned denominator. The effective value of a rational is the numerator / demoninator
17		
18	ImagePaletteColor	ImagePaletteColor structure
19		
20	IntegerRect	A rect composed of integer values. Typically used to specify the source rect (in pixels) of interest from an image.
21		
22	Matrix	Represents a 3x3 matrix used for transformations in two-dimensional space. Because "Avalon" only allows affine transformations, the Matrix structure has six entries instead of nine.
23		
24	NamedStringIndexer	This class is a helper to implement named indexers for strings
25		

1		localized in multiple cultures.
2	PixelFormat	Pixel Format Definition for images and pixel-based surfaces
3		
4	Delegates	
5	GetPageEventHandler	delegate GetPageEventHandler
6	HitTestFilterDelegate	Delegate for hit tester to control whether to test against children of visual.
7	HitTestResultDelegate	Delegate for hit tester to control returning of hit information on visual.
8		
9		
10	The following tables list example members exposed by the	
11	System.Windows.Media.Animation namespace.	
12	Classes	
13		Any class that doesn't derive from
14	Animatable	DependencyObject but which has properties that can be animated should derive from this class.
15		
16		This abstract class provides base
17	AnimationCollection	functionality for animation collections, such as ColorAnimationCollection, DoubleAnimationCollection, and SizeAnimationCollection.
18		
19		Override this class to implement
20	AnimationEffect	element level animations which can participate in the rendering process to instantiate animations on multiple elements at rendering time.
21		
22	AnimationEffectCollection	Holds a collection of AnimationEffects.
23	BoolAnimationCollection	Represents a collection of
24	BoolModifier	BoolModifier animations.
25	BoolTimedModifier	

1	ByteAnimationCollection	Represents a collection of BoolModifier animations.
2	ByteModifier	
3	ByteTimedModifier	
4	CharAnimationCollection	Represents a collection of CharModifier animations.
5	CharModifier	
6	CharTimedModifier	
7	ColorAnimation	Animates a color value of a property.
8	ColorAnimationCollection	Represents a collection of ColorModifier animations.
9	ColorKeyFrameCollection	
10	ColorModifier	
11	ColorTimedModifier	
12	DecimalAnimationCollection	Represents a collection of DecimalModifier animations.
13	DecimalModifier	
14	DecimalTimedModifier	
15	DoubleAnimation	Used to animate properties that accept a Double value.
16	DoubleAnimationCollection	Represents a collection of DoubleModifier animations.
17	DoubleKeyFrameCollection	
18	DoubleModifier	
19	DoubleTimedModifier	
20	FloatAnimation	Used to animate properties that accept a Single value.
21	FloatAnimationCollection	Represents a collection of FloatModifier animations.
22	FloatKeyFrameCollection	
23	FloatModifier	
24	FloatTimedModifier	
25	IntAnimationCollection	Represents a collection of IntModifier animations.
	IntModifier	
	IntTimedModifier	
	LengthAnimation	Used to animate properties that

1		accept a Length value.
2	LengthAnimationCollection	Represents a collection of
3	LengthKeyFrameCollection	LengthModifier animations.
4	LengthModifier	
5	LengthTimedModifier	
6	LongAnimationCollection	Represents a collection of
7	LongModifier	LongModifier animations.
8	LongTimedModifier	
9	MatrixAnimationCollection	Represents a collection of
10	MatrixModifier	MatrixModifier animations.
11	MatrixTimedModifier	
12	Modifier	
13	ObjectAnimationCollection	Represents a collection of
14	ObjectModifier	ObjectModifier animations.
15	ObjectTimedModifier	
16	PathAnimation	This animation can be used inside of
17	PointAnimation	a MatrixAnimationCollection to move a
18	PointAnimationCollection	visual object along a path.
19	PointKeyFrameCollection	Used to animate properties that
20	PointModifier	accept Point values.
21	PointTimedModifier	Represents a collection of
22	RectAnimation	PointModifier animations.
23	RectAnimationCollection	
24	RectKeyFrameCollection	Used to animate properties that
25	RectModifier	accept a Rect value.
	RectTimedModifier	Represents a collection of
	ShortAnimationCollection	RectModifier animations.
		Represents a collection of
		ShortModifier animations.

1	ShortModifier	
2	ShortTimedModifier	
3	SizeAnimation	Defines an animation based on the Size of an object. By providing Size information, an object can appear to shrink or enlarge over a period of time.
4	SizeAnimationCollection	Represents a collection of SizeModifier animations.
5	SizeKeyFrameCollection	
6	SizeModifier	
7	SizeTimedModifier	
8	StringAnimationCollection	Represents a collection of StringModifier animations.
9	StringModifier	
10	StringTimedModifier	
11	Timeline	Maintains run-time timing state for timed objects.
12	TimelineBuilder	An object that can be used to create Timeline objects.
13	TimeManager	The object that controls an entire timing tree.
14	TimeSyncValueTypeConverter	An object that performs type conversions involving TimeSyncValue values.
15	TimeTypeConverter	An object that performs type conversions involving Time values.
16	VectorAnimation	Used to animate properties that accept a Vector value.
17	VectorAnimationCollection	Represents a collection of VectorModifier animations.
18	VectorKeyFrameCollection	
19	VectorModifier	
20	VectorTimedModifier	
21		
22		
23		
24	Interfaces	
25	IClock	Represents an object that can provide linear

1		time values.
2	IModifier	Defines the basic behavior of a modifier object. A modifier is an object that takes an object, called the <i>base value</i> , of a certain type and returns another object of the same type as its output.
3		
4	ITimingControl	Defines the behavior of timelines and timed objects.
5	ITimingControlBuilder	Represents an object that can build a timeline template.
6		
7		
8	Enumerations	
9	AnimationType	Describes the behavior of an animation.
10	CloneType	The types of clones that CloneCore may request.
11	InterpolationMethod	Describes how an animation calculates its output values.
12	KeyTimeType	The different types of KeyTimes
13	TimeEndSync	Values for the endSync attribute, which specifies how a container calculates its simple duration based on the children's durations.
14		
15	TimeFill	Specifies how the timeline behaves after it is no longer active.
16	TimeRestart	Values for the Timeline.Restart attribute.
17	TimeSeekOrigin	Indicates a timeline position; used to specify the behavior of the ITimingControl interface's Seek method by defining the position to which the offset is applied.
18		
19	TimeSyncBase	The event to synchronize a begin or end value to.
20		
21		
22	Structures	
23	ColorKeyFrame	
24	DoubleKeyFrame	
25	FloatKeyFrame	

1	KeySpline	This class is used to pass an array of key splines into the KeySplines property of an animation fragment.
2		
3	KeyTime	A KeyTime is use to specify when relative to the time of an animation that a KeyFrame takes place.
4	LengthKeyFrame	
5	PointKeyFrame	
6	RectKeyFrame	
7	SizeKeyFrame	
8	Time	A value representing time, with associated time arithmetic operations.
9	TimelineEnumerator	Enumerates items in an TimelineList collection.
10	TimeSyncValue	A value representing an absolute or relative begin or end time for a timeline.
11	VectorKeyFrame	

The following tables list example members exposed by the System.Windows.Media.TextFormatting namespace.

Classes

16	InlineObjectInfo	Provides measurement details for inline text objects. The formatting client passes this object as a parameter to the GetInlineObjectInfo method.
17		
18		TextFormatter is the "Avalon" text engine and provides services for formatting text and breaking text lines. TextFormatter can handle different text character formats and paragraph styles, and includes support for international text layout.
19	TextFormatter	
20		
21	TextHighlightBounds	Bounds of text range
22		
23	TextInfo	Represents information about a block of text in the client's text source character store.
24		
25	TextLine	Provides services to a line of text. Inherit from this class to implement services

1		that manipulate and format a line of text. This is an abstract class.
2	TextMarkerGeneratedContent	Generates line list marker output.
3		Defines the style and type of a
4	TextMarkerInfo	paragraph's list marker. The formatting client uses this class as a parameter to provide marker details to the
5		GetTextMarkerInfo method.
6	TextParagraphProperties	Represents properties that can change from one paragraph to the next, such as flow direction, alignment, or indentation.
7		Defines a sequence of characters that
8	TextRun	share a single property set. The formatting client provides TextRun details into this class when the TextFormatter passes it as a
9		parameter to the GetTextRun method.
10	TextRunBounds	Bounds of text run
11		Provides caching services to the
12	TextRunCache	TextFormatter object in order to improve performance.
13	TextRunClientData	Represents client information data associated with a TextRun.
14		Provides properties that can change
15	TextRunProperties	from one TextRun to another, such as typeface or foreground brush. This is an abstract class.
16		Provides typography properties for
17	TextRunTypographyProperties	TextRun. This client set of properties generates a set of features that are processed by the OpenType layout engine.
18		Provides character data and
19		formatting properties to the TextFormatter.
20		All access to the text in the TextSource is
21	TextSource	achieved through the GetTextRun method, which is designed to allow the client to
22		virtualize text in any way it chooses. This is an abstract class.
23		Provides description of text trimming
24	TextTrimmingInfo	characteristics. The formatting client fills trimming details into this class when the TextFormatter passes it as a parameter of
25		

the GetTextTrimmingInfo method.

Enumerations

TextParagraphFlags	Flags describing paragraph characteristics
TextRunCacheFlags	Kind of content in text run cache
TextRunType	Indicates the type of TextRun.

Structures

TextSourceCharacterIndex	TextSourceCharacterIndex represents a caret or character position in text.
--------------------------	--

Media Integration Layer

A Media Integration Layer (MIL) provides an API for developers or programmers to accomplish possibly complex composition effects within their applications in a straightforward manner, while leveraging the graphics processing unit in a manner that does not adversely impact normal application performance. One aspect of the MIL provides the ability to combine different media types (e.g., 2D, 3D, Video, Audio, text, imaging and so forth) and animate them together smoothly and seamlessly.

The MIL provides a graphics architecture for multi-stage composition and a programming model that allows for functional parity at the programmatic and scripted interfaces. An API and script allows the creation of a retained structure or scene description that is composited when rendered, yet includes areas that have a more immediate-mode nature.

1 Via the interfaces, the MIL provides access to a data structure for storing
2 visual information so that applications can take advantage of the graphics
3 capabilities provided by the computer hardware. The interfaces are based on an
4 element object model and a vector graphics markup language for using that
5 element object model in a manner that allows program code developers to
6 consistently interface with a scene graph data structure to produce graphics. The
7 data structure may also be used for either directly rendering or for "compiling" the
8 visual information so that it can be provided to a lower level graphics system for
9 fast composition and animation.

10 The vector graphics element object model generally corresponds to shape
11 elements and other elements including image and video elements that correlate
12 with a scene graph object model of the scene graph. Markup may be parsed into
13 data including elements in an element tree that is translated into the objects of a
14 scene graph data structure. Other markup may be translated directly into data and
15 calls that create the scene graph objects. The markup language provides distinct
16 ways to describe an element, including a simple string format or complex property
17 syntax, which may be named, enabling reuse in other locations in the markup.

18 An aspect of the MIL is the integration of animation and timing across the
19 API set, providing animation as an inherent base-level concept. To facilitate
20 smooth animation, the MIL provides a multiple-level graphics processing system
21 and method (e.g., of an operating system). One such multiple-level graphics
22 processing system comprises two components, including a tick-on-demand or
23 slow-tick high-level component, and a fast-tick (e.g., at the graphics hardware
24 frame refresh rate) low-level component. In general, the high-level, less frequent
25 component performs computationally intensive aspects of updating animation

1 parameters and traversing scene data structures, in order to pass simplified data
2 structures to the low-level component. The low-level component operates at a
3 higher frequency, such as the frame refresh rate of the graphics subsystem, to
4 process the data structures into constant output data for the graphics subsystem.
5 The low-level processing includes interpolating any parameter intervals as
6 necessary to obtain instantaneous values to render the scene for each frame of
7 animation.

8 Top level MIL objects include a visual tree, which is an object that contains
9 the main content to be drawn. Controls will derive from visuals of the tree
10 directly. Visuals are device and parent context independent. A render target is the
11 device to which the visual is drawn. This object (e.g., screen) may have its own
12 dirty or invalidation mechanism. Various render targets include a screen in a
13 window, a Printer, a Metafile, a Surface, and a "Sub-window" which is a part of
14 the scene that is drawn separately from the rest of the scene. Other drawing
15 related objects include a Visual Renderer, comprising an object that is configured
16 to draw a visual tree onto a render target, and a Display Scheduler object that
17 knows when to draw the visual tree on to the render target. A Time Manager is a
18 context object for a set of timing nodes, and is the object that the scheduler calls
19 tick on.

20 A Visual API is provided, which is essentially a starting point for drawing
21 via the media integration layer, and comprises multiple types of objects, including
22 a VisualManager object, which connects a Visual Tree to a medium. The different
23 types of VisualManagers (e.g., Screen, Printer, and Surface) are responsible for
24 rendering a Visual Tree to their particular medium. A visual is where the
25

1 programmer does the drawing; it is a node in the visual tree and provides a place
2 for a program to draw.

3 The DrawingContext APIs present a context-based programming model for
4 how to construct visual content that populates a Visual or are rendered to an
5 ImageData. DrawingContext classes are provided, as well as the classes and
6 entrypoints necessary to acquire a DrawingContext and enumerate the visual
7 content in a RetainedVisual/DrawingVisual.

8 To enable mutability, there is provided a single set of types that derive from
9 a common Changeable base class. Any type for which mutability is desired
10 change may derive from the Changeable class. For example, in a graphics
11 programming, the object model includes Brushes, Pens, Geometries,
12 FloatAnimations, GradientStops, Segments, and so forth. An IsChangeable
13 property specifies whether the changeable object can be modified or not,
14 depending on its current value, which defines a state.

15 A brush is an object that represents a method to fill a plane. In addition to
16 being able to fill a plane in an absolute way, brushes of the media integration layer
17 are also able to adapt how they fill the plane relative to the size of the object that
18 they are filling. Examples of types of brushes include SolidColorBrush,
19 VisualBrush (which can reference a vector graphics resource/Visual),
20 DrawingBrush, LinearGradient, RadialGradient, ImageBrush and NineGridBrush.
21 Certain brush objects will have an idea of how they relate to the coordinate system
22 when they are used, and an idea of how they relate to the bounding box of the
23 geometry with which they are used. This size is based on from the object that the
24 brush is filling. The Brush base class has a Transform, a general opacity, and a
25 blend mode. Brush (and other object resources in Vector Graphics and the MIL

1 API) objects are Changeables and are writable after they have been created, and
2 follow the general Changeable pattern for how they behave after they are used in
3 qualified use.

4 A Geometry class of objects can be used for clipping, hit-testing and
5 rendering of 2D vector-based data with the Pen and Brush. The derived Geometry
6 classes provide more specific building and enumeration semantics. A number of
7 shape-specific Geometry types are provided, as well as a generalized
8 PathGeometry that allows for explicit definition of more complex shaped
9 Geometry. Geometry is an abstract base class. A GeometryCollection is a
10 collection of multiple Geometry objects that have been combined using particular
11 CombineMode operations on their defined area. This object allows easier building
12 visual combinations of Geometry objects than is possible using strictly PathFigure
13 objects within a PathGeometry.

14 ImageSource is an abstract class, comprising a basic building block for
15 imaging. An ImageSource conceptually represents a single, constant set of pixels
16 at a certain size and resolution. For example, an ImageSource may be a single
17 frame in an image file that a Decoder could provide, or it may be the results of a
18 transform that operates on a certain ImageSource of its own. An ImageSource is
19 changeable, not because its own properties can be changed, but because the
20 properties of its sub-classes can potentially be changed.

21 A Transform class of objects is provided for scaling, rotating, translating
22 and skewing vector and raster graphics. The derived Transform classes provide
23 friendly usage and enumeration semantics.

24 Effects provide a means to alter the visual contents of a scene in a
25 rendering-centric manner. For example, VisualEffects (raster-based bitmap

1 effects) operate on the image-based, fully composited representation of a portion
2 of a scene. Effects are broken down into various types including VisualEffects,
3 BlendModes and VectorEffects. VisualEffects can be used in a retained-mode
4 scene by applying it to a sub-graph or an Element, or it can be used in the
5 standalone image pipeline. BlendModes are a specific form of image-based
6 effects, and can be applied to the retained mode scene in generally the same
7 manner as VisualEffects. Blend modes perform a combination of the source and
8 destination colors as the source is composited, e.g., multiply or add.

9 Hit testing is used to pick visuals in a scene, and operates by starting from
10 the top of the control tree, and returning a control or set of controls by a point or
11 geometry. A control can define whether it is hit or not with support services
12 including rendered geometry, bounding box, out-of-band geometry (hit region),
13 image opacity or mask, and its own logic. The control can return specific hit-
14 related data on hit. The hit test mechanism can filter hit test results in an efficient
15 manner. The hit test walk is a deep right to left walk of the visual tree, with hits
16 reported through a callback in z-order, top-to-bottom fashion. When descending,
17 the hit tester views the filtering in terms of element level relationships, for
18 example, a canvas with shapes, or a dock panel with an inner canvas. When a hit
19 occurs, the hit tester can either continue processing further hits (if any), or stop.

20 An animation system is provided, comprised of a timing control engine and
21 a set of animation objects. The timing engine is a service that can be used by any
22 objects that exhibit time-varying behaviors, e.g., animations and audio or video
23 media objects. Animation objects implement a set of functions that map time
24 spans into other data types, which are then used as inputs into other higher-level
25 objects. Graphical animation is achieved by associating an animation collection

1 with a rendering operation. Each animation used in a rendering operation may be
2 run on a separate clock, referred to as a called a “timeline.” Once an animated
3 primitive is drawn and animation parameters specified, the low-level rendering
4 system takes care of redrawing the scene at regular intervals. Each time a frame is
5 rendered the current value of the animations involved in the scene is computed,
6 based on the elapsed time (in most cases measured by the system clock), and then
7 the animated primitives are redrawn.

8 Various primitive types, color features and media support is also provided
9 via the MIL. MediaData can be used to play any audio/video content.

12 Design Namespace

13 A design namespace 320 provides classes that enable the editing of forms
14 and text, formatting data and cross-process data sharing. These classes provide an
15 extensible framework for editing documents, applications, and other content.
16 Design namespace 320 contains functionality on two levels: high-level
17 functionality for application developers desiring pre-packaged ready-to-use editors
18 for different types of information; and lower-level functionality for more advanced
19 applications introducing its own types of data. Example pre-packaged ready-to-
20 use editors offer plain text editing, rich text editing, forms (element layout)
21 editing, and password input.

22 Design namespace 320 provides a flexible and extensible approach to
23 organizing various functionality. Instead of providing a common fixed-feature
24 editor, design namespace 320 provides a combination of services, behaviors and
25 abstractions that allow developers to build specialized and easily-customizable

1 information editing solutions. Design namespace 320 includes several
2 customizable editing features, including: scoped services, stackable behaviors,
3 editor-designer pattern, abstract designer for element editing, abstract text object
4 model, adorning layer, design surface, generalized data transfer protocol, and
5 extensible undo mechanism.

6 Scoped services allows the association of specific services with particular
7 portions and sub-portions of application data. This mechanism also allows turning
8 services on and off in different scopes. The main class providing a support for
9 services is ServiceManager; individual services are implemented using IService
10 and IScopedService interfaces.

11 The concept of "stackable behaviors" allows different behaviors to be
12 activated in appropriate periods of time. In one embodiment, time-based behavior
13 activation is nested, such that a sub-process can be started and finished while a
14 more general process is temporarily suspended and then properly restored. Rather
15 than hard-coding a solution, the approach of stackable behavior solves this
16 problem by allowing an integration of processes that do not have pre-defined
17 knowledge about each other. The main class providing support for this
18 mechanism is EditRouter; individual behaviors are subclasses of Editor and
19 EditBehavior classes. Built-in types of editable data are supported by
20 corresponding subclasses of EditBehavior such as TextEditor, ElementEditor and
21 MoveBehavior.

22 The concept of "editor-designer pattern" allows separation between generic
23 editing functionality and more specific sub-typed of editable information. The
24 editor does not expect a specific data structure on which to operate. Instead, it
25 assumes that data content is exposed through some abstract interface, hiding the

1 implementation details, but sufficient for corresponding action of editing
2 interaction. This pattern is particularly useful with elements (in forms editing) and
3 for text (in rich text editing).

4 "Abstract designer for element editing" is an application of the "editor-
5 designer pattern" for an area of element editing (e.g., forms editing). Various
6 elements may have different internal structures, appearances and behaviors, but if
7 they expose some standard "designers" for movement, rotation, or resizing, then
8 they become editable by the element editor.

9 The abstract text model is another application of the "editor-designer
10 pattern" for use in rich text editing. This abstraction allows different backing
11 stores to participate in rich text editing, such as RTF, XHTML, plain text, XAML
12 markup, syntactically highlighted source code, etc. Even though the semantic
13 nature and internal structure of these types of texts can be different, by exposing
14 itself via an abstract text model they allow the application of generic editing
15 mechanisms. Additionally, this approach allows for text and data integration, such
16 that text from different backing stores may contain pieces of each other, while
17 providing a seamless editing experience.

18 The adorning layer is an unified tool that provides rich visual feedback
19 during editing. The adorning layer provides uniformed support for many types of
20 feedback. Powerful visuals may be dynamically attached and detached to
21 elements and other portions of data to indicate potential user activity applicable to
22 them. The design surface is a XAML control that pre-packages editing resources,
23 such as the adorning layer and edit router. This simplifies middle-level editing
24 development.
25

Editing functions such as cut/copy/paste and drag/drop are important in an integrated information environment, but can be difficult to support due to the complexity and incompatibility of different data types. The generalized data transfer protocol addresses this difficulty by providing for the combination of abstractions used for data extraction, data insertion, data dragging, and data conversion. This generalized data transfer protocol provides a powerful and flexible mechanism for application-data integration.

The extensible undo mechanism manages an undo stack by collecting data of various types. The undo manager includes protocols to handle undo operations with different data types.

The following tables list example members exposed by the System.Windows.Design namespace.

Classes

ActivationEventFilter	EventFilter
ActivationRectangleAdornerInfo	This is the feedback that shows the border on DesignActive elements
AdornerInfo	Provides information about a specific adorer in an adornerset.
AdornerLayer	Provides a surface for displaying decorations on elements that need to transcend Z-order (the elements always need to be on top).
CanvasDesigner	Default Canvas designer.
ComponentChangeService	Provides undo and redo functionality for design-time actions that affect components.
Designer	Provides a means of editing live content.
DesignerFilterService	Defines an optional service that applications and controls can implement to override the default mapping between

1		an element and the designer provided by the DesignerLookupService.
2	DesignerLookupService	Allows edit behaviors to map elements to designers.
3		The recommended root element for
4	DesignSurface	editable content. It aggregates adorners, services, and events.
5	EditBehavior	Defines the implementation for all edit behaviors.
6		Handles user input for a particular
7	Editor	selection type, such as text, element, or table. Typically there is a specific editor for each selection type.
8		Associates a selection type with the selection mode. This class cannot be inherited.
9	EditorTypeAttribute	
10		Manages the set of currently active EditBehaviors.
11	EditRouter	
12	EditRouterChangedEventArgs	Arguments for Changed event
13	ElementEditor	Element editor
14	EventFilter	Enables event filtering and assignment.
15	FrameworkElementDesigner	The default Designer for all controls that derive from FrameworkElement.
16	MoveBehavior	Move Behavior
17	ObjectSelection	Site Selection
18	ParentUndoUnit	ParentUndoUnit
19	RichTextDesigner	ITextDesigner implementation for TextPanel and other controls that support ITextDocumentResult.
20		RoutedEventAdapter thunks a specific eventHandler down to an RoutedEventHandler so that one handler can handle different types of events. Used by EditRouter
21	RoutedEventAdapter	
22		SelectionRouter controls mixed selection with multiple selection modes. It derives from EditRouter to route events to modular editors (which derive from
23		
24	SelectionRouter	
25		

1		Editor).
2	SelectionService	Provides programmatic access to selection of UI items. You can get feedback on selected items, as well as change selected items and their properties.
3		
4	ServiceConverter	ServiceConverter - Converter class for converting between a string and an instance of IService.
5		
6	SimpleTextDesigner	ITextDesigner implementation for Text and other controls that support ITextParagraphResult.
7		
8		TextDecorator defines an object that manages text decoration from competing services. Using TextDecorator instead of writing property values privately ensures that competing decoration will never overlap, even in the case of identical priority.
9	TextDecorator	
10		
11	TextEditor	Provides text selection services.
12		
13	TextSelection	The TextSelection class encapsulates selection state for the TextEditor class. It has no public constructor, but is exposed via a public property on TextEditor.
14		
15	UndoService	Framework implementation of IUndoService
16		
17	UndoStackChangedEventArgs	Provides data for the UndoStackChanged and RedoStackChanged events
18		
19		
20	Interfaces	
21		
22	IAddRemoveChildDesigner	The IAddRemoveChildDesigner interface is used for adding/removing children
23		
24	IAdornerDesigner	Provides access to methods that allow you to retrieve the objects necessary to draw adorners on FrameworkElements.
25		

1		The inner element in
2	IDesignActivationContainer	DesignActivation that can become
3		designActive. Has the name container as
4		it is assumed that if an element is
5	IDesignActivationContainerParent	DesignActive, then it will most likely be
6		editing some child content
7	IDesignSurface	An
8		IDesignActivationContainerParent can
9	IMoveDesigner	be host IDesignActivationContainer's in
10	IParentUndoUnit	a Design environment
11	IScopedService	Interface to be implemented by
12	ISelection	any element wishing to participate in
13		editing.
14		Move designer interface
15		IParentUndoUnit interface
16		IScopedService
17		base interface for selection
18		modes
19		Designer interface associated
20		with the SelectionBehavior
21		(SelectionRouter). A class that
22		implements this would be specified in
23		the Designer Attribute of an element
24		type
25		This is a placeholder for a
26		TypeConverter for editing services.
27		An interface components
28		implement to supply the TextEditor
29		service with layout-specific information
30		and actions.
31		IUndoService interface
32		IUndoUnit interface
33		
34		
35		
36		
37		
38		
39		
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
100		
101		
102		
103		
104		
105		
106		
107		
108		
109		
110		
111		
112		
113		
114		
115		
116		
117		
118		
119		
120		
121		
122		
123		
124		
125		
126		
127		
128		
129		
130		
131		
132		
133		
134		
135		
136		
137		
138		
139		
140		
141		
142		
143		
144		
145		
146		
147		
148		
149		
150		
151		
152		
153		
154		
155		
156		
157		
158		
159		
160		
161		
162		
163		
164		
165		
166		
167		
168		
169		
170		
171		
172		
173		
174		
175		
176		
177		
178		
179		
180		
181		
182		
183		
184		
185		
186		
187		
188		
189		
190		
191		
192		
193		
194		
195		
196		
197		
198		
199		
200		
201		
202		
203		
204		
205		
206		
207		
208		
209		
210		
211		
212		
213		
214		
215		
216		
217		
218		
219		
220		
221		
222		
223		
224		
225		
226		
227		
228		
229		
230		
231		
232		
233		
234		
235		
236		
237		
238		
239		
240		
241		
242		
243		
244		
245		
246		
247		
248		
249		
250		
251		
252		
253		
254		
255		
256		
257		
258		
259		
260		
261		
262		
263		
264		
265		
266		
267		
268		
269		
270		
271		
272		
273		
274		
275		
276		
277		
278		
279		
280		
281		
282		
283		
284		
285		
286		
287		
288		
289		
290		
291		
292		
293		
294		
295		
296		
297		
298		
299		
300		
301		
302		
303		
304		
305		
306		
307		
308		
309		
310		
311		
312		
313		
314		
315		
316		
317		
318		
319		
320		
321		
322		
323		
324		
325		
326		
327		
328		
329		
330		
331		
332		
333		
334		
335		
336		
337		
338		
339		
340		
341		
342		
343		
344		
345		
346		
347		
348		
349		
350		
351		
352		
353		
354		
355		
356		
357		
358		
359		
360		
361		
362		
363		
364		
365		
366		
367		
368		
369		
370		
371		
372		
373		
374		
375		
376		
377		
378		
379		
380		
381		
382		
383		
384		
385		
386		
387		
388		
389		
390		
391		
392		
393		
394		
395		
396		
397		
398		
399		
400		
401		
402		
403		
404		
405		
406		
407		
408		
409		
410		
411		
412		
413		
414		
415		
416		
417		
418		
419		
420		
421		
422		
423		
424		
425		
426		
427		
428		
429		
430		
431		
432		
433		
434		
435		
436		
437		
438		
439		
440		
441		
442		
443		
444		
445		
446		
447		
448		
449		
450		
451		
452		
453		
454		
455		
456		
457		
458		
459		
460		
461		
462		
463		
464		
465		
466		
467		
468		
469		
470		
471		
472		
473		
474		
475		
476		
477		
478		
479		
480		
481		
482		
483		
484		
485		
486		
487		
488		
489		
490		
491		
492		
493		
494		
495		
496		
497		
498		
499		
500		
501		
502		
503		
504		
505		
506		
507		
508		
509		
510		
511		
512		
513		
514		
515		
516		
517		
518		
519		
520		
521		
522		
523		
524		
525		
526		
527		
528		
529		
530		
531		
532		
533		
534		
535		
536		
537		
538		
539		
540		
541		
542		
543		
544		
545		
546		
547		
548		
549		
550		
551		
552		
553		
554		
555		
556		
557		
558		

1	EditRouterChange	Enumeration of possible router change actions
2	TextDecorationPriority	TextDecorationPriority ranks the relative importance of various text markup. It is used to z-order decoration runs in the TextDecorator class.
3	UndoState	Enum for the state of the undo manager
4		
5		
6	Delegates	
7		Represents the method that will
8	UndoStackChangedEventHandler	handle an UndoStackChanged or RedoStackChanged event.
9		
10		

Input Namespace

An input namespace 322 includes an input manager that coordinates inputs received by the system. The input namespace 322 also includes classes that help manage and provide control for different input devices, such as a keyboard or a mouse. The input system allows applications to get information from input devices such as mice, keyboards, and styluses (pens). Most input functionality is in the form of properties, methods, and events on the UIElement, FrameworkElement, ContentElement, and ContentFrameworkElement classes in the System.Windows namespace. UIElement and ContentElement have similar input functionality, which is captured by the IInputElement interface. Similarly, FrameworkElement and ContentFrameworkElement, which derive from UIElement and ContentElement respectively, share many input APIs and both implement IFrameworkInputElement.

1 The input system provides full support for keyboards, mouse, and styles.
2 Keyboard functionality includes keydown/up events, focus management and
3 notification, accessing current key state, and converting keystrokes into input
4 strings. Mouse functionality includes mouse position, clicks, movement,
5 enter/leave events at element boundaries, mouse capture, hover and mouse wheel.
6 Stylus functionality includes position and movement (both when the pen is
7 touching the surface and when it is "in air"), taps/clicks, capture, and gesture
8 recognition.

9 The Input namespace contains helper classes necessary for the above
10 functionality, such as enumerations, events arguments, delegate signatures, etc.
11 Additionally, the Input namespace includes the Keyboard, Mouse, and Stylus
12 classes, which provide information relating to the current state of those devices.
13 The input system provides ways of specifying mouse cursors on a particular
14 element, and for an entire application.

15 The input system is integrated with Text Services Framework (TSF), which
16 allows other input software to understand the textual context into which new text
17 is being input. This is used by input method editors (IMEs), which allow users of
18 East Asian languages to turn multiple keystrokes into a single character -- the IME
19 software communicates with TSF to identify the best interpretation of the
20 keystrokes based on context and uses TSF to insert the text into the document.
21 Similarly, speech recognition uses TSF to pick the best recognition and insert it
22 into the document.

23 The input system offers automatic integration with TSF, in the form of a
24 TextInput event. The Input namespace also offers APIs for describing the textual
25 context, and controlling the conversion from raw keyboard input into textual input.

The InputManager class provides filters and monitors, which allow a third party to observe the input stream and create new input events before the original input events have been fired. The InputManager also provides APIs for new devices to be plugged into the input system, using IInputProvider and InputReport.

The following tables list example members exposed by the System.Windows.Input namespace.

Classes

AccessKeyManager

AccessKeyManager exposes access key functionality, also known as mnemonics. A typical example is the Ok button, with alt-O being the mnemonic – pressing alt-O is the same as clicking the button.

Cursor

Represents the mouse cursor to use.

CursorTypeConverter

Converts mouse cursors to and from strings.

FocusChangedEventArgs

The FocusChangedEventArgs class contains information about focus changes.

InputDevice

Provides the base class for all input devices.

InputEventArgs

The InputEventArgs class represents a type of RoutedEventArgs that are relevant to all input events.

InputLanguageChangedEventArgs

The InputLanguageEventArgs class represents a type of RoutedEventArgs that are relevant to events raised to indicate a change in (human) input language.

InputLanguageChangingEventArgs

The InputLanguageEventArgs class represents a type of RoutedEventArgs that are relevant to events raised to indicate a change in

1		(human) input language.
2	InputLanguageEventArgs	The InputLanguageEventArgs class represents a type of RoutedEventArgs that are relevant to events raised to indicate a change in (human) input language.
3		
4	InputLanguageManager	The InputLanguageManager class is responsible for managing the input language in Avalon.
5		
6	InputManager	The InputManager class is responsible for coordinating all of the input system in "Avalon".
7		
8	InputMethod	The InputMethod class exposes TSF-related APIs, which are communicating or accessing TIP's properties.
9		
10	InputMethodStateChangedEventArgs	This InputMethodStateChangedEventArgs class is used when the input method editor (IME) changes its state.
11		
12	InputProviderSite	The object which input providers use to report input to the input manager.
13		
14	InputReport	The InputReport is an abstract base class for all input that is reported to the InputManager.
15		
16	InputReportEventArgs	Provides data for the event that is raised when an InputReport is being processed.
17		
18	Keyboard	The Keyboard class exposes APIs related to the keyboard.
19		
20	KeyboardDevice	The KeyboardDevice class represents the keyboard device to the members of a context.
21		
22	KeyboardEventArgs	The KeyboardEventArgs class provides access to the logical pointer device for all derived event args.
23		
24	KeyEventArgs	The KeyEventArgs class contains information about key states.
25	KeyInterop	Provides static methods to

1		convert between Win32 VirtualKeys and the “Avalon” Key enum.
2	Mouse	The Mouse class exposes APIs related to the mouse.
3	MouseButtonEventArgs	The MouseButtonEventArgs describes the state of a Mouse button.
4		
5	MouseDevice	The MouseDevice class represents the mouse device to the members of a context.
6		
7	MouseDoubleClickEventArgs	Provides data for events that are raised when the mouse is doubled-clicked.
8		
9	MouseEventArgs	The MouseEventArgs class provides access to the logical Mouse device for all derived event args.
10	MouseWheelEventArgs	The MouseWheelEventArgs describes the state of a Mouse wheel.
11		
12	NotifyInputEventArgs	Provides information about an input event being processed by the input manager.
13	PreProcessInputEventArgs	Allows the handler to cancel the processing of an input event.
14	ProcessInputEventArgs	Provides access to the input manager's staging area.
15		
16	RawKeyboardInputReport	The RawKeyboardInputReport class encapsulates the raw input provided from a keyboard.
17		
18	RawMouseInputReport	The RawMouseInputReport class encapsulates the raw input provided from a mouse.
19		
20	StagingAreaInputItem	This class encapsulates an input event while it is being processed by the input manager.
21		
22	TextInputEventArgs	The TextInputEventArgs class contains a text representation of input.
23		
24	TextManager	The TextManager class provides the input-to-text event promotion.
25		

TextServicesContext	This class manages interop between a <code>UIDispatcher</code> and the Text Services Framework, a native COM API that enables east-asian IME input.
---------------------	---

TextStoreInfo This is an internal, link demand protected class.

Interfaces

InputLanguageSource An interface for controlling the input language source.

IInputProvider	An interface implemented by all input providers.
-----------------------	--

IKeyboardInputProvider	An interface for controlling the keyboard input provider.
-------------------------------	---

IMouseInputProvider	An interface for controlling the mouse input provider.
---------------------	--

Enumerations

CursorType	An enumeration of the supported cursor types.
------------	---

ImeConversionMode ImeConversionMode

ImeSentenceMode ImeSentenceMode

InputMethodState	State of Ime
------------------	--------------

InputMode	The mode of input processing when the input was provided.
-----------	---

InputType	The type of input being reported.
-----------	-----------------------------------

Key	An enumeration of all of the possible key values on a keyboard.
-----	---

KeyState The **KeyState** enumeration describes the state that keyboard keys can be in.

ModifierKeys The **ModifierKeys** enumeration describes a set of common keys used to modify other input operations.

1	MouseButton	The MouseButton enumeration describes the buttons available on the mouse device.
2	MouseButtonState	The MouseButtonState enumeration describes the possible states of the buttons available on the Mouse input device.
3		
4	RawKeyboardActions	The raw actions being reported from the keyboard.
5	RawMouseActions	The raw actions being reported from the mouse.
6	SpeechMode	Mode of speech
7		
8		
9	Structures	
10	TextServicesMSG	Managed version of the Win32 MSG struct.
11		
12	Delegates	
13		
14	FocusChangedEventHandler	The delegate to use for handlers that receive FocusChangedEventArgs.
15		
16	InputEventHandler	The delegate to use for handlers that receive InputEventArgs.
17		
18	InputLanguageEventHandler	This is a delegate for InputLanguageChanged and InputLanguageChanging events.
19		
20	InputMethodStateChangedEventHandler	The delegate to use for handlers that receive input method state changed event.
21		
22	InputReportEventHandler	The delegate to use for handlers that receive PointerMoveEventArgs.
23	KeyboardEventHandler	The delegate to use for handlers that receive KeyboardEventArgs.
24		
25	KeyEventHandler	The delegate to use for handlers that receive

		EventArgs.
		The delegate to use for
1		handlers that receive
2	MouseButtonEventHandler	MouseButtonEventArgs.
3		The delegate to use for
4	MouseDoubleClickEventHandler	handlers that receive
5		MouseDoubleClickEventArgs.
6	MouseEventHandler	The delegate to use for
7		handlers that receive
8	MouseWheelEventHandler	MouseEventArgs.
9		The delegate to use for
10	NotifyInputEventHandler	handlers that receive
11		MouseWheelEventArgs.
12	PreProcessInputEventHandler	Delegate type for handles
13		of events that use
14	ProcessInputEventHandler	NotifyInputEventArgs.
15		Delegate type for handles
16	TextInputEventHandler	of events that use
17		PreProcessInputEventArgs.
18		Delegate type for handles
19		of events that use
20		ProcessInputEventArgs.
21		The delegate to use for
22		handlers that receive
23		TextInputEventArgs.
24		
25		

Navigation Namespace

A navigation namespace 324 provides a set of classes and services that allow the building of applications with navigation paradigms, such as a browser application. These classes and services permit the development of applications with customized navigation experiences. For example, when purchasing a product or service from an online merchant, clicking a “Back” button causes the application to display a different page that asks the user if they want to cancel or change their order. In another example, activating a “Refresh” button causes an

application to retrieve new data instead of first reloading the application followed by retrieving the new data. The navigation namespace 324 also includes page functions that provide a mechanism for generating a hierarchy of questions that are presented to a user.

The following tables list example members exposed by the System.Windows.Navigation namespace.

Classes

BoolWrapper	A wrapper for a Boolean value.
BoolWrapperPageFunction	A typed page function that returns a Boolean value to the previous page.
BoolWrapperReturnEventArgs	
IntWrapper	A wrapper for a Int32 value.
IntWrapperPageFunction	A typed page function that returns a Int32 value to the previous page.
IntWrapperReturnEventArgs	
Journal	Contains an application's navigation history.
JournalEntry	Represents a journal entry.
LoaderService	Used to set the current loader in a given appdomain.
NavigateEventArgs	Obsolete.
NavigatingCancelEventArgs	Event arguments for the NavigationApplication.Navigating and NavigationWindow.Navigating events.
NavigatingNewWindowCancelEventArgs	Event args for the NavigatingNewWindow cancelable event The NavigatingNewWindowCancelEventArgs specifies the target NavigationContainer where the

1		navigation will take place with the
2		Uri or element passed in. By default
3		the Cancel property is set to false.
4		Setting Cancel to true will prevent the
5	NavigationApplication	new window from being opened, and
6		the navigation will not take place.
7		Represents a "Longhorn"
8	NavigationContainer	navigation application to the system.
9		A navigable region that can
10		contain an "Avalon" markup tree.
11		Normally, this class is not used
12		directly but can be used as the parent
13	NavigationErrorCancelEventArgs	class for a custom implementation.
14		Contains the arguments for the
15		NavigationApplication.NavigationErr
16		or and
17		NavigationWindow.NavigationError
18		events.
19		Event arguments for non-
20	NavigationEventArgs	cancellable navigation events,
21		including LoadCompleted,
22		LoadStarted, Navigated, and
23		NavigationStopped.
24		Contains the arguments for the
25	NavigationProgressEventArgs	NavigationApplication.NavigationPro
		cess and
		NavigationWindow.NavigationProces
		s events.
		Contains the delegates used by
	NavigationService	navigation events, and a dynamic
		property that contains an INavigator
		interface.
		Represents a navigation
	NavigationWindow	window.
		A typed page function that
	ObjectPageFunction	returns an Object value to the
		previous page.
	ObjectReturnEventArgs	
		This class is not directly
	PageFunction	supported. Instead use one of the
		typed classes:

1		BoolWrapperPageFunction,
2		IntWrapperPageFunction,
3		ObjectPageFunction, or
4		StringPageFunction.
5	ReturnArgs	The event argument object for the Return event. This class is not directly supported. Instead use the return arguments for the appropriate typed class:
6		BoolWrapperReturnEventArgs,
7		IntWrapperReturnEventArgs,
8		ObjectReturnEventArgs, or
9	StringPageFunction	StringReturnEventArgs.
10	StringReturnEventArgs	A typed page function that returns a String value to the previous page.
11	WindowNavigationContainer	Represents a navigable region within a navigation window.
12		
13		
14	Interfaces	
15	IJournalData	IJournalData interface - Should to be implemented by Controls that need to persist state in the journal, and restore it when the page is revisited
16	ILoader	Interface used to resolve Uri's to streams. It can be used to load content from file, http, container and managed and unmanaged resources
17	INavigator	Implemented by navigation containers to provide access to the properties, methods, and events that support navigation.
18	INavigatorService	INavigatorService interface. This interface will be available on the NavigationWindow enabling any implementor of INavigator to register themselves to participate in Hyperlink targeting.
19		
20		
21		
22		
23		
24		
25	Enumerations	

1	NavigationMode	Used to specify the navigation mode.
2		
3	Delegates	
4	BoolWrapperReturnEventHandler	
5	IntWrapperReturnEventHandler	
6		Represents the method that
7	LoadCompletedEventHandler	handles the NavigationApplication.LoadComple ed and NavigationWindow.LoadCompleted events.
8		
9	LoadStartedEventHandler	Represents the method that handles the LoadStarted event.
10		Represents the method that
11	NavigatedEventHandler	handles the NavigationApplication.Navigated and NavigationWindow.Navigated events.
12		
13	NavigateEventHandler	Obsolete.
14		Represents the method that
15	NavigatingCancelEventHandler	handles the NavigationApplication.Navigating and NavigationWindow.Navigating events.
16		
17	NavigatingNewWindowCancelEvent	Represents the method that
18	Handler	handles the NavigatingNewWindow event.
19		This delegate is used with the
20	NavigationErrorCancelEventHandler	NavigationApplication.NavigationEr ror and NavigationWindow.NavigationError events.
21		
22		This delegate is used with the
23	NavigationProgressEventHandler	NavigationWindow.NavigationProgr ess and NavigationApplication.NavigationPr ogress events.
24		
25	NavigationStoppedEventHandler	Represents the method that

handles the
NavigationApplication.NavigationSt
opped and
NavigationWindow.NavigationStopp
ed events.

ObjectReturnEventHandler

Represents the method that
handles the Return event. This class
is not directly supported. Instead use
the return event handler for the
appropriate typed class:
BoolWrapperReturnEventHandler,
IntWrapperReturnEventHandler,
ObjectReturnEventHandler, or
StringReturnEventHandler.

ReturnEventHandler

StringReturnEventHandler

Automation Namespace

An automation namespace 326 contains members used to support
accessibility and user interface automation. The accessibility system includes a
client side and a provider side. A system of tools includes client side automation
tools including a client automation class for seeking user interface information.
The client automation class includes event registration tools and logical element
discovery tools. The set of tools further includes provider side automation tools
for providing the client with user interface information. The provider side
automation tools include an automation provider class having tools for providing
the client with event information.

A client automation class provides UI automation methods for one or more
clients. The client automation class contains methods that are not specific to any
UI element. The client automation class may provide a method for obtaining a

logical or raw element from a point, a window handle, or the desktop root element. The client automation class additionally may provide methods for finding a logical element based input criteria. The client automation class preferably also includes a method for registering and un-registering for event notifications. The automation class preferably also provides helper functions for loading proxy DLLs, retrieving the localized names of properties and control patterns and performing element comparisons. The client automation class also includes methods for clients to listen for events. The following tables list example members exposed by the System.Windows.Automation namespace.

Classes

ApplicationWindowPattern

Exposes the behavior and information typically associated with a top-level application window. Clients can use this class to tile or cascade the application's multiple document interface (MDI) children, find its button on the taskbar, and locate well-known sections of its user interface such as toolbars and menus.

AssociatedInformationPattern

Exposes semantic and metadata for UI elements that represent other objects.

Automation

Contains User Interface (UI) Automation methods for clients (assistive technology or automated test script). These methods are not specific to a particular element.

AutomationEvent

Do not use. This class will be removed in future versions.

AutomationEventArgs

Pattern or custom event args class

1	AutomationFocusChangedEventArgs	Focus event args class
2	AutomationIdentifier	Base class for object identity-based identifiers. This class is effectively abstract; only derived classes are instantiated.
3		
4	AutomationIdentifierProxy	Internal class used to deserialize AutomationIdentifiers. Should not be used directly.
5		
6	AutomationPattern	Do not use. This class will be removed in future versions.
7	AutomationPermission	Provides a set of permissions for accessing UI elements. This class cannot be inherited.
8		
9	AutomationPermissionAttribute	Provides methods and properties for AutomationPermissionAttribute. This class cannot be inherited.
10		
11	AutomationProperty	Do not use. This class will be removed in future versions.
12	AutomationPropertyChangedEventArgs	PropertyChanged event args class
13	S	
14	AutomationTextAttribute	Identifier for Automation Text Attributes
15		Represents the position of a character within text.
16	AutomationTextPointer	AutomationTextPointer provides methods and properties for accessing text and text navigation.
17		
18	AutomationTextRange	Used to get, set, add and remove selection.
19		
20		Purpose: The AutomationTextSelection object handles all text selection management. The selection that the insertion pointer is on is the active selection. Example usages: It is used when clients want to add, remove, modify or set selection. Clients can also find out what is currently selected through AutomationTextSelection.
21	AutomationTextSelection	
22		
23		
24		
25		

1	AutomationTextUnit	Identifier for Automation Text Units
2	BasePattern	Internal class
3	DockPattern	Expose an element's ability to change its dock state at run time.
4		This exception is thrown
5	ElementNotEnabledException	when client code attempts to manipulate an element or control that is currently not enabled.
6		ElementPath provides
7		criteria necessary to subsequently
8	ElementPath	return to the logical element previously recorded, modified, or wholly created by the application provider.
9		
10		Exposes a control's ability to
11		expand to display more content or
12	ExpandCollapsePattern	to collapse to hide content. Examples include menu buttons, Start button, tree-view item in Windows Explorer, and combo boxes.
13		
14		Exposes the elements of a
15	GridItemPattern	grid. Allows clients to quickly determine the coordinates of a grid item.
16		
17	GridPattern	Exposes basic grid functionality, size, and navigation to specified cells.
18		
19		Exposes the hierarchical
20	HierarchyItemPattern	relationship between a control's user interface elements independent of their relationship in the logical tree.
21		
22	Input	Provides methods for sending mouse and keyboard input
23		
24	InvokePattern	Implemented by objects (controls) that perform a single, unambiguous action. Most of these controls do not maintain state; invoking them initiates an action in
25		

1		the application. Examples of
2		controls that implement this
3		interface include push buttons,
4		toolbar buttons, menu items,
5		hyperlinks, check boxes, radio
6	LogicalElement	buttons, the plus symbol in the tree-
7		view control, and list-view items in
8	LogicalStructureChangedEventArgs	Windows Explorer.
9		Represents a UI element in
10		the logical tree regardless of
11		implementation ("Avalon",
12	MultipleViewPattern	Microsoft® Win32®).
13		Logical structure changed
14		event args class.
15		Exposes an elements's ability
16		to switch between multiple
17	NoClickablePointException	representations of the same set of
18		information, data, or children.
19		Examples include ListView
20		(Thumbnails, Tiles, Icons, List,
21		Details) Excel charts (Pie, Line,
22	ProxyAssemblyNotLoadedException	Bar, Cell Value with a formula),
23		and Outlook Calendar (Year,
24		Month, Week, Day).
25	RangeValuePattern	The exception that is thrown
		when the value of an argument is
		outside of the allowable range of
		values, as defined by
		GetClickablePoint. For example,
		when the bounding rectangle is
		empty, has no width or heighth or the
		LogicalElement at that point is not
		the same one that was called.
		This exception is thrown
		when there is a problem laoding a
		proxy assembly. This can happen In
		reponse to
		Automation.RegisterProxyAssembl
		y or when loading the default
		proxies when the first hwnd base
		LogicalElement is encountered.
		Exposes a control's ability to

1		manage a value within a finite range. It conveys a control's valid minimum and maximum values and current value.
2		
3	RawElement	Represents an element in the raw element tree.
4		
5	ScrollPattern	Represents UI elements that are expressing a value
6		
7		Represents individual items in containers that manage selection. UI Automation clients use this class to get information about, or to manipulate, UI elements that support the SelectionItemPattern control pattern. Examples include any items that can be selected in a control that supports the SelectionPattern control pattern, such as an item in a list box or tree view.
8	SelectionItemPattern	
9		
10		
11		
12		
13		Represents containers that manage selection. Used by UI Automation clients to get information about, or to manipulate, user interface elements that support the SelectionPattern control pattern.
14	SelectionPattern	
15		
16		Exposes a container's current sort order and allows clients to programmatically re-sort elements.
17	SortPattern	
18		
19	SplitPattern	Represents windows that can clone themselves by creating an adjacent window.
20		
21	TableItemPattern	Represents grid items with header information.
22	TablePattern	Represents a grid that has header information.
23	TextPattern	Represents text, such as an edit control.
24	TopLevelWindowEventArgs	TopLevelWindowEventArgs event args class
25	TreeLoadEventArgs	TreeLoadEventArgs event

1		args class
2	ValuePattern	Exposes values for controls that do not span a range such as radio buttons, toggle buttons, check boxes, edit boxes, RGB color values, and checkable menu items.
3		
4		Exposes information about an image or an animation that conveys information to the user such as Internet Explorer's waving flag or spinning globe that indicates a document in downloading or Windows Explorer's flying papers that indicate copying is in progress.
5	VisualInformationPattern	
6		
7		Exposes an element's ability to change its on-screen position or size. Examples include top-level application windows (Word or Windows Explorer), Outlook's main window panes (folders, e-mail messages, tasks), and MDI child windows.
8	WindowPattern	
9		
10		Exposes the current zoom level in a control and allows clients to change it programmatically.
11	ZoomPattern	
12		
13		
14		
15		
16		
17	Enumerations	
18		
19	AutomationPermissionFlag	Contains the access flags for an AutomationPermission object.
20	ChildArrangement	Different ways children may be arranged
21	DockPosition	The edge of the container that the dockable window will cling to.
22	ExpandCollapseState	Used by ExpandCollapse pattern to indicate expanded/collapsed state
23	GetDataOptions	Options for GetData
24	HierarchyItemState	Enumeration that indicates the state of a hierarchy item: collapsed, expanded, or partially expanded
25		

1	ItemCheckState	Enumeration that indicates state of checkbox, radio buttons, and similar controls
2	LoadState	Tree state flags.
3	LogicalMapping	Values that indicate whether a node in the raw tree will be displayed in the logical tree
4	ProxyFlags	Enum used to indicate results of requesting a property
5	RowOrColumnMajor	Is the data data in this table best present by row or column
6	ScopeFlags	Flags used to define scope when listening to events
8	ScrollAmount	Used by ScrollPattern to indicate direction (forward or backward) and amount (page or line) by which to scroll
10	SendMouseInputFlags	Flags for SendMouseInput. These flags indicate whether movementnt took place, or whether buttons were pressed or released.
12	SplitStyles	Flag for the directions into which a window may split
13	StructureChangeType	Flags that indicate changes in the structure of the logical tree.
15	TextPointerGravity	Gravity determines the placement of a AutomationTextPointer when text is inserted at the AutomationTextPointer. Gravity is always from the text pointers point of view, i.e. I am before my character or I am after my character. When the user inserts a character at your text pointer location, Before means that your pointer will be placed before the new character; After means that the pointer will be placed After the new character.
20	WindowChangeType	Flags that indicate changes in top-level windows.
21	WindowInteractionState	The current state of the window for user interaction
23	WindowVisualState	States used to describe the visual state of the window. WindowVisualState follows the Office and HTML definition of WindowState.
24		
25		

Structures

MatchCondition	Specifies criteria for finding elements using FindLogicalElement or FindRawElement.
NameAndData	Contains name, data as an object, and data as a string
ProxyDescription	Structure containing information about a proxy
SortInformation	Information used to sort

Delegates

AutomationEventHandler	Delegate to handle AutomationEvents
AutomationFocusChangedEventHandler	Delegate to handle focus change events
AutomationPropertyChangedEventHandler	Delegate to handle Automation Property change events
LogicalStructureChangedEventHandler	Delegate to handle logical structure change events
ProxyFactoryCallback	Implemented by HWND handlers; called by UI Automation framework to request a proxy for specified window. Should return a proxy if supported, or null if not supported.
TopLevelWindowEventHandler	Delegate to handle top level window events

The following tables list example members exposed by the System.Windows.Automation.InteropProvider namespace.

Classes

1		Contains methods used by Win32
2	AutomationInteropProvider	applications or controls that implement User Interface (UI) Automation. This class cannot be inherited.
3		
4	TextPointerPair	Represent a contiguous block of characters
5		
6	Interfaces	
7		
8	IApplicationWindowInteropProvider	Exposes the behavior and information typically associated with a top-level application window.
9		
10	IAssociatedInformationInteropProvider	Exposes semantic and metadata for UI elements that represent other objects.
11		
12	IDockInteropProvider	Expose an element's ability to change its dock state at run time.
13		
14		Exposes a control's ability to expand to display more content or collapse to hide content. Supported in conjunction with the HierarchyItem pattern on TreeView items to provide tree-like behavior, but is also relevant for individual controls that open and close. Examples of UI that implements this includes: - TreeView items - Office's smart menus that have been collapsed - Chevrons on toolbars - Combo box - Menus - "Expandos" in the task pane of Windows Explorer (left-hand side where folder view is often displayed).
15		
16		
17		
18	IExpandCollapseInteropProvider	
19		
20		
21		
22		
23		
24		
25	IGridInteropProvider	Exposes basic grid functionality: size and moving

1		to specified cells.
2	IGridItemInteropProvider	Represents an item that is within a grid. Has no methods, only properties.
3		Expose and allow clients to traverse the hierarchical relationship between UI elements independent from their relationship in teh logical tree. Hierarchy relationships are by definition noncircular.
4		Examples of UI that implements this includes: -
5		TreeView items - Visio Org
6	IHierarchyItemInteropProvider	Chart - Menus - Listview controls when "Show in groups" mode is active .
7		Implemented by objects (controls) that perform a single, unambiguous action. Most of these controls do not maintain state; invoking them initiates an action in the application.
8		Examples of user interface elements that implement this interface include push buttons, toolbar buttons, menu items, hyperlinks, check boxes, radio buttons, the plus symbol in the tree-view control, and list-view items in Windows Explorer.
9	IInvokeInteropProvider	Exposes an element's ability to switch between multiple representations of the same set of information, data, or children This pattern should be implemented on the container which controls the current view of content.
10		Exposes a related set of
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22	IMultipleViewInteropProvider	
23		
24		
25	IRangeValueInteropProvider	

1		properties that reflect a controls
2		ability to manage a value within
3		a finite range. It conveys a
4		controls valid minimum and
5		maximum values and its current
6		value. Examples: Numeric
7		Spinners Progress Bar, IP
8		Control (on the individual
9		octets) some Color Pickers
10		ScrollBars some Sliders public
11		interface that represents UI
12		elements that are expressing a
13		current value and a value range.
14		public interface has same
15		definition as IValueProvider.
16	IRawElementProviderFragment	The two patterns' difference is
17		that RangeValue has additional
18		properties, and properties
19		generally do not appear in the
20		pattern public interfaces.
21		Implemented by ,
22	IRawElementProviderFragmentRoot	providers to expose elements
23		that are part of a structure more
24		than one level deep. For simple
25	IRawElementProviderFragmentRootAdviseEvents	one-level structures which have
		no children,
		IRawElementProviderSimple
		can be used instead. The root
		node of the fragment should
		support the
		IRawElementProviderFragment
		Root interface, which is derived
		from this, and has some
		additional methods.
		The root element in a
		fragment of UI should support
		this interface. Other elements in
		the same fragment need to
		support the
		IRawElementProviderFragment
		interface.
		Implemented on the root
		element of a UI fragment to

1		allow it to be notified of when it is required to raise automation events.
2		Implemented by
3		providers which want to
4	IRawElementProviderHwndOverride	provide information about or
5		want to reposition contained
6		HWND-based elements.
7		Implemented by
8		providers which want to
9		provide information about or
10		want to reposition contained
11	IRawElementProviderSimple	HWND-based elements.
12		UIAutomation provider
13		interface, implemented by
14		providers that want to expose
15		properties for a single element.
16		To expose properties and
17		structure for more than a single
18		element, see the derived
19	IScrollInteropProvider	IRawElementProviderFragment
20		interface
21		The Scroll pattern
22		exposes a control's ability to
23		change the portion of its visible
24		region that is visible to the user
25		by scrolling its content.
		Examples: Listboxes
		TreeViews other containers that
		maintain a content area larger
		than the control's visible region
		Note that scrollbars themselves
		should not support the
		Scrollable pattern; they support
		the RangeValue pattern.
		Servers should normalize
		scrolling (0 to 100). This public
		interface represents UI
		elements that scroll their
		content.
		Provides access to the
	ISelectionInteropProvider	properties implemented by non-

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

ISelectionItemInteropProvider

"Longhorn" providers on containers that manage selection.

Provides access to methods and properties that define and manipulate selected items in a container. Is only supported on logical elements that are children of elements that supports ISelectionInteropProvider and is itself selectable.

ISortInteropProvider

Expose a container's current sort order and allow clients to programmatically re-sort its elements. Some containers maintain the sort order when inserting new items, or move their contents based on updated content (example: sorted Win32 listboxes). Other containers are only capable of performing a one-time sort that becomes out-of-date when new items are inserted (example: Excel).

ISplitInteropProvider

Exposes the ability to split an elements content area into multiple panes or interactive areas.

ITableInteropProvider

Identifies a grid that has header information.

ITableItemInteropProvider

Used to expose grid items with header information.

ITextInteropProvider

The Text pattern exposes a control's ability to manipulate text. Examples: TextBox RichEdit controls other containers that contain text and text related properties This interface represents UI elements that maintain text.

The Text pattern exposes a control's ability to manipulate text. Examples: TextBox RichEdit controls other containers that contain text and text related properties This interface represents UI elements that maintain text.

public interface that
represents UI elements that are
expressing a value

Used to express information about an image or an animation that is conveying information to the user.

Expose an element's ability to change its on-screen position or size, as well as change the visual state and close it.

Exposes the current zoom level in a control and allows clients to programmatically change it.

Directions for navigation the UIAutomation tree

Indicates whether this provider

1 A serialization namespace 328 provides a parser that can load or save a
2 hierarchy of objects (e.g., elements) from or to an XML file or a file with a binary
3 representation. This process also sets properties associated with the objects and
4 associates event handlers. Further, serialization namespace 328 provides
5 functionality for serializing and de-serializing objects. Serialization is a process of
6 taking a live, in-memory representation of objects, and producing a stream of data
7 suitable for saving to a storage device (such as a disk) or sending across a
8 network. Deserialization is the reverse process; i.e., taking a data stream and
9 producing objects from the data stream. Deserialization is also referred to as
10 "parsing", and the code that implements parsing is known as the parser.

11 Both serialization and deserialization support multiple streaming formats,
12 such as an XML format referred to as XAML or a binary format referred to as
13 BAML. When the parser is used with a markup compiler, it can generate
14 "CAML", where the output is code to generate the serialized objects.

15 Serialization and deserialization support the full range of XAML features,
16 including: standard syntax, compound properties, compact compound syntax,
17 implicit child syntax, explicit children syntax, collection syntax with explicit
18 collection tag, and collection syntax with implicit collection tag. Both
19 serialization and deserialization are fully extensible. Classes can define how they
20 wish to be serialized and deserialized through several mechanisms, such as:

- 21 • By default, class names are treated as markup tag names, and properties and
22 events are treated as markup attributes of the same name.
- 23
24
25

- Classes define type converters (auxiliary objects driving from `System.ComponentModel.TypeConverter`) to serialize and deserialize markup attribute values.
- Classes can support implicit children by implementing the `IAddChild` interface (parsing) and using CLR attributes to describe how to serialize implicit children.
- Classes can control whether a property can be serialized/deserialized by using the `DesignerSerializationVisibility`, and control whether it is necessary to serialize a property on a particular object by using the `DefaultValueAttribute` and `ShouldPersist` methods.
- Or, a component can serialize/deserialize itself using literal content, in which case the component's serialized form does not have to conform to the rules of XAML (the component needs to be well-formed XML).

Although the above examples use the word "markup", the same concepts apply to the binary format (BAML).

A second extensibility mechanism is "designer hooks", which allows another piece of code to plug into the serializer and deserializer, and insert and intercept attributes during parsing and serialization. This mechanism allows new markup features to be created, and allows custom processing of existing syntax. The markup compiler uses designer hooks to generate CAML.

Serialization and deserialization support a variety of objects (e.g., any class driving from `System.Object`). Classes driving from

System.Windows.DependencyObject can define dependency properties, which get two extra features:

- Serialization can automatically tell whether a property needs to be serialized (no need for DefaultValueAttribute/ShouldPersist)
- The parser can perform certain optimizations to load faster

Serialization and deserialization support XML namespaces and definition files that map XML namespaces to CLR namespaces. The Mapper class allows the definition of arbitrary mappings from XML namespaces and tag names into CLR classes.

The following tables list example members exposed by the System.Windows.Serialization namespace.

Classes

CustomXamlDeserializer	The base class for the deserialization helper. This should be subclassed if design or edit time deserialization direction of the parser is required.
CustomXamlSerializer	The base class for the serialization helper. This should be subclassed if design or edit time serialization direction of the serializer is required.
DependencyPropertyContext	Context information about the current property being serialized. The XamlSerializer is responsible for putting an instance of this class on the context stack before serializing any property on an object.
DPSerializationVisibilityAttribute	Specifies the visibility of this property or method as seen by the

1		designer serializer.
2	EntityContext	Abstract base class for ObjectContext and PropertyContext.
3	LayoutAttribute	Attribute for specifying the LayoutType for a tag.
4		Handles mapping between Extensible Markup Language (XML) namespace Uniform Resource Identifier (URI) and Microsoft® .NET namespace types.
5	Mapper	
6		Contains information the Mapper uses for Mapping between an xml NamespaceUri and what Assembly, urtNamespace to look in.
7	NamespaceMap	
8		Context information about the current object being serialized. The XamlSerializer is responsible for putting an instance of this class on the context stack in SerializeObject.
9	ObjectContext	
10		"Avalon" Element Parsing class used to Create an "Avalon" Tree.
11	Parser	
12		Provides all the context information required by Parser
13	ParserContext	
14		Context information about the current property being serialized. The XamlSerializer is responsible for putting an instance of this class on the context stack before serializing any property on an object.
15	PropertyContext	
16		Context information about the current object being serialized. The XamlSerializer is responsible for putting an instance of this class on the context stack in SerializeObject.
17	TextContext	
18		XamlAttributeNode
19	XamlAttributeNode	
20		XamlClrArrayPropertyNode
21	XamlClrArrayPropertyNode	
22		XamlClrComplexPropertyNode, which is an object specified as the child of a clr object
23	XamlClrComplexPropertyNode	
24		XamlClrEventNode, which is a clr event on any object. Note that this
25	XamlClrEventNode	

1		may be on a DependencyObject, or any other object type.
2	XamlClrObjectNode	XamlClrObjectNode, which is anything other than a DependencyObject
3		XamlClrPropertyNode, which is a
4	XamlClrPropertyNode	clr property on any object. Note that this may be on a DependencyObject, or any other object type.
5		Exception class for parser
6	XamlClrPropertyParseException	property parsing specific exceptions
7		XamlComplexDependencyPropertyNode, which is a DependencyProperty
8	XamlComplexDependencyPropertyNode	specified as the xml child of a DependencyObject
9		XamlComplexPropertySerializer
10	XamlComplexPropertySerializer	is used to serialize properties that can not be serialized as XML attributes. ComplexProperties are instead serialized as markup within the element.
11		
12	XamlDefAttributeNode	XamlDefAttributeNode
13	XamlDefTagNode	XamlDefTagNode
14	XamlDependencyObjectSerializer	Extends XamlSerializer to include animations (specific to Elements)
15		XamlDependencyPropertyNode, which is a normal, simple
16	XamlDependencyPropertyNode	DependencyProperty on a DependencyObject
17		Exception class for parser
18	XamlDependencyPropertyParseException	dynamic property parsing specific exceptions
19		
20	XamlDesignerSerializerAttribute	Specifies the type name of serializer for the given class.
21	XamlDocumentNode	XamlDocument Node
22		XamlElementNode, which represents a DependencyObject. These are different from CLR objects because they can have Dependency Properties associated with them.
23	XamlElementNode	
24		
25	XamlEndAttributesNode	XamlEndAttributesNode

1	XamlEndClrArrayPropertyNode	XamlEndClrArrayPropertyNode
2	de XamlEndClrComplexPropertyNo	XamlEndClrComplexPropertyNo
3	XamlEndClrObjectNode	XamlEndClrObjectNode
4	opertyNode XamlEndComplexDependencyPr	XamlEndComplexDependencyPr
5	XamlEndDocumentNode	XamlEndDocumentNode
6	XamlEndElementNode	XamlEndElementNode
7	e XamlEndIDictionaryPropertyNod	XamlEndIDictionaryPropertyNod
8	XamlEndIListPropertyNode	XamlEndIListPropertyNode
9	XamlEndResourceNode	XamlEndResourceNode
10	XamlEndStartElementNode	XamlEndStartElementNode
11	XamlIDictionaryPropertyNode	XamlIDictionaryPropertyNode
12	XamlIEnumeratorSerializer	Serializes the node collection pointed to by the property.
13	XamlIListPropertyNode	XamlIListPropertyNode
14	XamlIListSerializer	Serializes the IList pointed to by the property.
15	XamlIncludeTagNode	XamlIncludeTagNode
16	XamlLanguageNode	XamlLanguageNode
17	XamlLiteralContentNode	XamlLiteralContentNode
18	XamlNode	Base Node in which all others derive
19	XamlParseException	Exception class for parser specific exceptions
20	XamlPIMappingNode	XamlPIMappingNode which maps an xml namespace to a clr namespace and assembly
21	XamlResourceNode	XamlResourceNode
22	XamlRootSerializer	XamlSerializer is used to persist logical tree.
23	XamlRoutedEventNode	XamlRoutedEventNode
24	XamlSerializationCallbackExcept	Exception class for serialization callback specific exceptions
25	XamlSerializer	XamlSerializer is used to persist the logical tree.
	XamlSerializerBase	Base class for XamlSerializer

1		providing common helper functions.
2	XamlTextNode	XamlTextNode
3	XamlTextNodeSerializer	Overrides XamlSerializer to special case TextNodes.
4	XamlXmlnsPropertyNode	XamlXmlnsPropertyNode
5	XmlAttributes	A class to encapsulate XML-specific attributes of a DependencyObject
6	XmlnsDictionary	A dictionary to control XML namespace mappings
7	XmlParserDefaults	Public class used by Avalon.
8		
9		
10	Interfaces	
11	IAddChild	The IAddChild interface is used for parsing objects that allow objects or text underneath their tags in markup that do not map directly to a property.
12	IBamlSerialize	Serialization interface for Dynamic Properties written to BAML files
13		
14	ILoaded	This interface provides a read-only boolean property called IsLoaded, a clr event handler called Loaded, and DeferLoad and EndDeferLoad methods
15		
16	IPageConnector	Provides methods used internally by the BamlReader on compiled content.
17	IParseLiteralContent	For Literal Content - content which has its own parser and saver.
18		
19	IUriContext	The IUriContext interface allows elements (like Frame, PageViewer) and type converters (like ImageData TypeConverters) a way to ensure that base uri is set on them by the parser, codegen for xaml, baml and caml cases. The elements can then use this base uri to navigate.
20		
21		
22		
23	Enumerations	
24		
25	LayoutType	Layout Types that can be associated with an Object.

1		Describes the action the serializer or
2	SerializationAction	deserializer is to take after it has called back to the CustomXamlSerializer or CustomXamlDeserializer helpers
3		Describes the action the serializer or
4	SerializationErrorAction	deserializer is to take when an error has been reported
5	XamlNodeType	Identifier for XamlNodes

Interop Namespace

An interop namespace 330 provides a set of classes that enable interoperability with other operating systems or computing platforms. The following tables list example members exposed by the System.Windows.Interop namespace.

Classes

14		MarshalByRefObject wrapper over
15	ApplicationProxy	Application class to allow interaction with Application object across AppDomains and to allow Application creation on a different thread.
16		Interop class used for implementing the
17	DocobjHost	managed part of a Docobj Server for browser hosting
18	PresentationInteropHelper	VisualInteropHelper
19		Implements Avalon
20	WindowInteropHelper	WindowInteropHelper classes, which helps interop b/w legacy and Avalon Window.

Delegates

24	AppEntryPoint	Delegate for the Application code's entry point method.
----	---------------	---

Forms.Interop Namespace

A forms.interop namespace 332 provides an element that allows an application to host a form control operation.

System.IO.CompoundFile Namespace

Another namespace, System.IO.CompoundFile (not shown in Fig. 3) provides services to utilize a compound file in which various document distributable files are stored. These services allow for the encryption and compression of content. The services also support the storage of multiple renditions of the same content, such as a re-flowable document and a fixed-format document. The following tables list example members exposed by the System.IO.CompoundFile namespace.

Classes

CertificateHelper

Helper to get a digital certificate for signing

CompoundFileByteRangeReference

Substream reference component that refers to a range of bytes within a compound file stream.

CompoundFileIndexReference

Substream reference component that refers to a logical entry within a compound file stream.

CompoundFileReference

A reference to a portion of a compound file.

CompoundFileReferenceCollection

A read-only collection of

1		compound file references.
2	CompoundFileStorageReference	Logical reference to a container storage
3	CompoundFileStreamReference	Logical reference to a container stream
4	CompressionTransform	CompressionTransform for use in Compound File DataSpaces
5		This class enables users to get access to the StorageRoot of the current container based on the loaded Application instance
6	ContainerHelper	Used to manipulate the data spaces within a specific instance of the "Avalon" container. This is how data transform modules are plugged into the container to enable features like data compression and data encryption.
7		Read-only class that enables clients to inspect and validate existing digital signatures.
8	DigitalSignature	
9		A read-only collection of digital signatures.
10	DataManager	
11		This class is used to create, persist, and manipulate digital signatures in a compound file.
12	DigitalSignatureCollection	
13		Contains property elements corresponding to the properties in the standard OLE document summary information property set.
14	DigitalSignatureManager	
15		Class for manipulating version object
16	DocumentSummaryInfo	The exception that is thrown when the format of the DrmTransformInstanceData on disk is invalid.
17		
18	FormatVersion	
19		
20	InstanceDataFormatException	
21		
22	OleProperty	
23		
24		
25		

1	PropSet	
2		Class for manipulating
3	RenditionInfo	information of a rendition. This
4		class also provides APIs for
5	RenditionInfoCollection	adding and removing storages and
6		streams to and from the rendition.
7	RenditionManager	
8		Class for manipulating
9	RightsManagementEncryptionTransform	renditions in a container
10		Class implements
11	StorageInfo	IDataTransform interface, so it
12		can be used as a Container data
13	StorageRoot	transform that implements RM
14		encryption and decryption of the
15	StreamInfo	content
16		Class for manipulating
17	SummaryInfo	storages in the container file
18		Represents the main
19	TransformEnvironment	container class. There is one
20		instance of the StorageRoot per
21	TransformInitializationEventArgs	compound file.
22		Provides access for
23	UseLicenseInfo	manipulating streams in the
24		container file.
25		

1		describing a use license returned from the Tungsten server.
2	UseLicenseInfoEnumerator	Represents an enumerator over the use licenses stored in the DRM Transform's instance data.
3		
4	VersionTuple	Class for a version tuple which consists of major and minor numbers
5		
6	XmlDigitalSignatureProcessor	Signature processor used by the digital signature manager to sign and validate data according to the W3C recommendation.
7		
8		
9		
10	Interfaces	
11		
12	IDataTransform	Interface to be implemented by all data transform objects
13		This interface is used to decouple RMTransform and RMWizard by building custom implementation of this interface we enable 3rd parties to take advantage of the RMWizard without forcing them to use RMTransform
14	ILicenseStorage	
15		
16	ISignatureProcessor	Signature processor interface
17	IUnknownRCW	This interface is used to opaquely handle any COM interface (wrapped in the Runtime COM Wrapper) as an IUnknown.
18		
19		
20	Enumerations	
21	DigitalSignatureProcessor	Predefined Digital Signature Processors
22	SignatureProcessorNameType	Type of signature Processor
23		When naming a transform object, the string being passed in can be interpreted in one of several ways. This enumerated type is used to specify the semantics of the
24	TransformIdentifierTypes	
25		

identification string.

Delegates

InvalidSignatureHandler Called with signatures that failed
hash verification

TransformInitializeEventHandler Delegate method for initializing
transforms

System.Windows.Automation Namespace

A System.Windows.Automation namespace contains members used to support accessibility and user interface automation. The accessibility system includes a client side and a provider side. A system of tools includes client side automation tools including a client automation class for seeking user interface information. The client automation class includes event registration tools and logical element discovery tools. The set of tools further includes provider side automation tools for providing the client with user interface information. The provider side automation tools include an automation provider class having tools for providing the client with event information.

A client automation class provides UI automation methods for one or more clients. The client automation class contains methods that are not specific to any UI element. The client automation class may provide a method for obtaining a logical or raw element from a point, a window handle, or the desktop root element. The client automation class additionally may provide methods for finding a logical element based input criteria. The client automation class preferably also includes a method for registering and un-registering for event notifications. The automation class preferably also provides helper functions for loading proxy DLLs, retrieving

the localized names of properties and control patterns and performing element comparisons. The client automation class also includes methods for clients to listen for events. The following tables list example members exposed by the System.Windows.Automation namespace.

Classes

ApplicationWindowPattern	Exposes the behavior and information typically associated with a top-level application window. Clients can use this class to tile or cascade the application's multiple document interface (MDI) children, find its button on the taskbar, and locate well-known sections of its user interface such as toolbars and menus.
AssociatedInformationPattern	Exposes semantic and metadata for UI elements that represent other objects.
Automation	Contains User Interface (UI) Automation methods for clients (assistive technology or automated test script). These methods are not specific to a particular element.
AutomationEvent	Do not use. This class will be removed in future versions.
AutomationEventArgs	Pattern or custom event args class
AutomationFocusChangedEventArgs	Focus event args class
AutomationIdentifier	Base class for object identity-based identifiers. This class is effectively abstract; only derived classes are instantiated.
AutomationIdentifierProxy	Internal class used to deserialize AutomationIdentifiers.

1		Should not be used directly.
2	AutomationPattern	Do not use. This class will be removed in future versions.
3	AutomationPermission	Provides a set of permissions for accessing UI elements. This class cannot be inherited.
4		Provides methods and properties for AutomationPermissionAttribute. This class cannot be inherited.
5	AutomationPermissionAttribute	
6		Do not use. This class will be removed in future versions.
7	AutomationProperty	
8	AutomationPropertyChangedEventArgs	PropertyChanged event args class
9	S AutomationTextAttribute	Identifier for Automation Text Attributes
10		Represents the position of a character within text. AutomationTextPointer provides methods and properties for accessing text and text navigation.
11	AutomationTextPointer	
12		Used to get, set, add and remove selection.
13	AutomationTextRange	
14		Purpose: The AutomationTextSelection object handles all text selection management. The selection that the insertion pointer is on is the active selection. Example usages: It is used when clients want to add, remove, modify or set selection. Clients can also find out what is currently selected through AutomationTextSelection.
15		
16	AutomationTextSelection	
17		Identifier for Automation Text Units
18	AutomationTextUnit	
19	BasePattern	Internal class
20	DockPattern	Expose an element's ability to change its dock state at run time.
21		This exception is thrown when client code attempts to
22	ElementNotEnabledException	
23		
24		
25		

1		manipulate an element or control that is currently not enabled.
2		ElementPath provides
3	ElementPath	criteria necessary to subsequently return to the logical element
4		previously recorded, modified, or wholly created by the application
5		provider.
6		Exposes a control's ability to expand to display more content or
7	ExpandCollapsePattern	to collapse to hide content. Examples include menu buttons,
8		Start button, tree-view item in Windows Explorer, and combo
9		boxes.
10		Exposes the elements of a grid. Allows clients to quickly
11	GridItemPattern	determine the coordinates of a grid item.
12		Exposes basic grid
13	GridPattern	functionality, size, and navigation to specified cells.
14		Exposes the hierarchical
15	HierarchyItemPattern	relationship between a control's user interface elements independent
16		of their relationship in the logical tree.
17	Input	Provides methods for sending mouse and keyboard input
18		Implemented by objects
19		(controls) that perform a single, unambiguous action. Most of these
20		controls do not maintain state; invoking them initiates an action in
21	InvokePattern	the application. Examples of controls that implement this
22		interface include push buttons, toolbar buttons, menu items,
23		hyperlinks, check boxes, radio buttons, the plus symbol in the tree-
24		view control, and list-view items in Windows Explorer.
25		

1		Represents a UI element in
2	LogicalElement	the logical tree regardless of
3		implementation ("Avalon",
4		Microsoft® Win32®).
5		Logical structure changed
6	LogicalStructureChangedEventArgs	event args class.
7		Exposes an elements's ability
8		to switch between multiple
9		representations of the same set of
10		information, data, or children.
11		Examples include ListView
12	MultipleViewPattern	(Thumbnails, Tiles, Icons, List,
13		Details) Excel charts (Pie, Line,
14		Bar, Cell Value with a formula),
15		and Outlook Calendar (Year,
16		Month, Week, Day).
17		The exception that is thrown
18		when the value of an argument is
19		outside of the allowable range of
20		values, as defined by
21	NoClickablePointException	GetClickablePoint. For example,
22		when the bounding rectangle is
23		empty, has no width or height or the
24		LogicalElement at that point is not
25		the same one that was called.
		This exception is thrown
		when there is a problem loading a
		proxy assembly. This can happen in
		response to
	ProxyAssemblyNotLoadedException	Automation.RegisterProxyAssembly
		or when loading the default
		proxies when the first hwnd base
		LogicalElement is encountered.
		Exposes a control's ability to
		manage a value within a finite
		range. It conveys a control's valid
		minimum and maximum values and
		current value.
	RangeValuePattern	
		Represents an element in the
	RawElement	raw element tree.
		Represents UI elements that
	ScrollPattern	are expressing a value

1		Represents individual items
2		in containers that manage selection.
3		UI Automation clients use this class
4	SelectedItemPattern	to get information about, or to
5		manipulate, UI elements that
6		support the SelectedItemPattern
7		control pattern. Examples include
8		any items that can be selected in a
9		control that supports the
10		SelectionPattern control pattern,
11		such as an item in a list box or tree
12		view.
13		Represents containers that
14		manage selection. Used by UI
15	SelectionPattern	Automation clients to get
16		information about, or to manipulate,
17		user interface elements that support
18		the SelectionPattern control pattern.
19		Exposes a container's current
20	SortPattern	sort order and allows clients to
21		programmatically re-sort elements.
22		Represents windows that can
23	SplitPattern	clone themselves by creating an
24		adjacent window.
25		Represents grid items with
26	TableItemPattern	header information.
27		Represents a grid that has
28	TablePattern	header information.
29		Represents text, such as an
30	TextPattern	edit control.
31		TopLevelWindowEventArgs
32	TopLevelWindowEventArgs	event args class
33		TreeLoadEventArgs event
34	TreeLoadEventArgs	args class
35		Exposes values for controls
36		that do not span a range such as
37		radio buttons, toggle buttons, check
38		boxes, edit boxes, RGB color
39		values, and checkable menu items.
40		Exposes information about
41	VisualInformationPattern	an image or an animation that

1		conveys information to the user
2		such as Internet Explorer's waving
3		flag or spinning globe that indicates
4		a document in downloading or
5		Windows Explorer's flying papers
6		that indicate copying is in progress.
7	WindowPattern	Exposes an element's ability
8		to change its on-screen position or
9		size. Examples include top-level
10		application windows (Word or
11		Windows Explorer), Outlook's
12		main window panes (folders, e-mail
13		messages, tasks), and MDI child
14		windows.
15	ZoomPattern	Exposes the current zoom
16		level in a control and allows clients
17		to change it programmatically.
18		
19	Enumerations	
20	AutomationPermissionFlag	Contains the access flags for an
21		AutomationPermission object.
22	ChildArrangement	Different ways children may be arranged
23	DockPosition	The edge of the container that the
24		dockable window will cling to.
25	ExpandCollapseState	Used by ExpandCollapse pattern to
26		indicate expanded/collapsed state
27	GetDataOptions	Options for GetData
28	HierarchyItemState	Enumeration that indicates the state of a
29		hierarchy item: collapsed, expanded, or
30		partially expanded
31	ItemCheckState	Enumeration that indicates state of
32		checkbox, radio buttons, and similar controls
33	LoadState	Tree state flags.
34	LogicalMapping	Values that indicate whether a node in
35		the raw tree will be displayed in the logical tree
36	ProxyFlags	Enum used to indicate results of
37		requesting a property

1	RowOrColumnMajor	Is the data data in this table best present by row or column
2	ScopeFlags	Flags used to define scope when listening to events
3	ScrollAmount	Used by ScrollPattern to indicate direction (forward or backward) and amount (page or line) by which to scroll
4	SendMouseInputFlags	Flags for SendMouseInput. These flags indicate whether movementnt took place, or whether buttons were pressed or released.
5	SplitStyles	Flag for the directions into which a window may split
6	StructureChangeType	Flags that indicate changes in the structure of the logical tree.
7	TextPointerGravity	Gravity determines the placement of a AutomationTextPointer when text is inserted at the AutomationTextPointer. Gravity is always from the text pointers point of view, i.e. I am before my character or I am after my character. When the user inserts a character at your text pointer location, Before means that your pointer will be placed before the new character; After means that the pointer will be placed After the new character.
8	WindowChangeType	Flags that indicate changes in top-level windows.
9	WindowInteractionState	The current state of the window for user interaction
10	WindowVisualState	States used to describe the visual state of the window. WindowVisualState follows the Office and HTML definition of WindowState.
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22	Structures	
23	MatchCondition	Specifies criteria for finding elements using FindLogicalElement or FindRawElement.
24	NameAndData	Contains name, data as an object, and data as a string
25		

ProxyDescription	Structure containing information about a proxy
SortInformation	Information used to sort

Delegates

AutomationEventHandler	Delegate to handle AutomationEvents
AutomationFocusChangedEventHandler	Delegate to handle focus change events
AutomationPropertyChangedEventHandler	Delegate to handle Automation Property change events
LogicalStructureChangedEventHandler	Delegate to handle logical structure change events
ProxyFactoryCallback	Implemented by HwndAutomation framework to request a proxy for specified window. Should return a proxy if supported, or null if not supported.
TopLevelWindowEventHandler	Delegate to handle top level window events

System.Windows.Ink Namespace

A System.Windows.Ink namespace provides classes that support electronic ink processing systems. Electronic ink processing techniques are useful to a variety of software applications. These electronic ink processing techniques are particularly applicable to the analysis of electronic ink, including layout analysis, classification, and recognition of electronic ink. Certain electronic ink processing techniques allow the electronic ink to be processed asynchronously with regard to

1 the operation of the software application implementing the techniques, so that the
2 electronic ink can be processed without stopping or significantly delaying the
3 operation of the software application. The software application can even continue
4 to accept new electronic ink input while previous electronic ink input is being
5 processed.

6 An application programming interface instantiates an ink analyzer object
7 that receives document data for a document containing electronic ink content from
8 a software application hosting the document and running on a first processing
9 thread. The ink analyzer object then employs the first thread to make a copy of
10 the document data, provides the copy of the document data to an electronic ink
11 analysis process, and returns control of the first processing thread to the analysis
12 process. After the analysis process has analyzed the electronic ink, the ink
13 analyzer object reconciles the results of the analysis process with current
14 document data for the document.

15 In particular embodiments, elements in a file or document may be described
16 based upon their spatial position relative to each other. For example, both an
17 electronic ink stroke and typewritten text may be described in terms of the same
18 spatial coordinate system. Using spatial information to describe the elements of a
19 document, the software application managing the document can maintain a data
20 structure describing the relationship between its document elements. In particular,
21 the software application can maintain a data structure both describing the class of
22 the various document elements and defining associations between the various
23 document elements. These associations can be defined, for example, as
24 information used to link electronic ink stroke data or collections thereof to other
25

elements in the electronic document (such as words, lines, paragraphs, drawings, table cells, etc.).

By describing document elements in a file or document data structure based upon their spatial position, document elements for a variety of file types can employ common techniques for identifying and manipulating their document elements. More particularly, a variety of software applications can describe document elements within a document based upon their spatial position and employ this spatial position referencing to use common electronic ink analysis methods. Still further, by specifying a particular region of a document for analysis, each software application can limit the analysis process to only desired elements within a document.

To analyze new electronic ink input into a document, the software application managing the document modifies a data structure associated with the document to include the new ink to be analyzed. The software application then provides this data structure (or relevant portions thereof) to an ink analysis tool, which copies some or all of the data structure for analysis (and operates on this copy of the data that is independent of the application program's document data structure). The ink analysis tool passes the copy to an analysis process, such as a parsing process (e.g., a layout analysis process and/or a classification process). The software application may resume its normal operation, including receiving new electronic ink input and/or other data, while the ink analysis process(es) is (are) being performed. In addition to receiving new electronic ink, any "other data" may be received by the application program, for example, data modifying a size, location, or content of existing ink, text, images, graphics, tables, flowcharts, diagrams, and the like; data adding additional text, images, graphics, tables,

1 flowcharts, diagrams, and the like; data deleting existing text, images, graphics,
2 tables, flowcharts, diagrams, and the like. After all desired analysis processes are
3 completed, the analysis results are returned to the ink analysis tool.

4 An ink analysis tool performs a variety of functions to facilitate the
5 processing of electronic ink. The ink analysis tool can be implemented as a
6 programming interface, such as an API. Further, the ink analysis tool can be
7 implemented as a group of software object routines and related information that
8 can be called by a software application as needed to analyze ink in a document.

9 In one embodiment, an API embodying an implementation of an ink
10 analysis tool (referred hereafter to as an Ink Analysis API), may contain two core
11 classes. The first class is referred to as the "Analysis Context" class, and the
12 second class is the "Ink Analyzer" class. The components of the "Analysis
13 Context" class are used to create an analysis context object. The components of
14 the "Ink Analyzer" class are then used to create and employ an object that
15 provides a document independent analysis context object to an analysis process,
16 determine when the analysis results have been generated, and reconcile the
17 analysis results with the current state of a document.

18 The Analysis Context class is implemented by a host application to create
19 an analysis context object, which serves as a proxy view onto the internal
20 document tree of the software application. The analysis context object contains all
21 unanalyzed ink data, and the analysis context object is used to identify which
22 unanalyzed ink data should be analyzed. The analysis context object also contains
23 information about previously analyzed ink. This previously analyzed ink may be
24 used to decide how the currently unanalyzed ink should be analyzed, and itself
25 may be modified in the course of analyzing the unanalyzed ink. Further, the

1 analysis content object contains information about non-ink content of the
2 document, which is used to properly classify ink as annotations to the non-ink
3 content.

4 The Analysis Context class includes a constructor which, when called by
5 the software application, creates the analysis context object. This class may also
6 include various properties for the analysis context object, including the property
7 entitled “Dirty Region {get;}”. The Dirty Region property defines the portion of
8 the document (and thus the portion of the analysis context object) that contains
9 unanalyzed ink data.

10 The Analysis Context class may also include a property entitled “Rootnode
11 {get;}”, which identifies the topmost or root context node in the analysis context
12 object. This root context node contains, as child context nodes, all other context
13 nodes objects for the given analysis context object. In particular embodiments, the
14 root context node should be of the context node type “Root.” In embodiments
15 where the application implements its own analysis context object, the analysis
16 context object may have other context nodes as siblings of the root context node,
17 but components of the Ink Analyzer class may be limited to considering context
18 nodes contained by the root context node.

19 The Analysis Context class may additionally include the property “Analysis
20 Hints {get;}”, which returns an array of analysis hint objects set by the software
21 application. As will be discussed in more detail below, the analysis hint objects
22 may contain any type of information that may assist the analysis process. This
23 information may include, for examples, as factoids, guides, or a word list. It may
24 also include information setting a language to be used for analysis, information
25 designating the unanalyzed ink as handwritten text only or drawing only, or

1 providing any kind of guidance to the parsing process, such as identifying the ink
2 as lists, tables, shapes, flow charts, connectors, containers and the like.

3 In addition to these properties, the Analysis Context class may also include
4 various methods that may be called by, *e.g.*, the software application to have the
5 analysis context object execute a task. For example, the Analysis Context class
6 may include a method entitled “FindNode (Guid id).” Each node in the analysis
7 context object has a globally unique identifier (or GUID), and this method will
8 locate the node specified in the call anywhere in the analysis context object.

9 Like the Analysis Context class, the Ink Analyzer class also defines a
10 public constructor that allows the software application to create an instance of the
11 class (*i.e.*, an ink analyzer object), along with various properties. For example, it
12 may contain a property entitled “User Interface Context {get; set;},” which defines
13 the processing thread to which the results of the analysis process are returned. This
14 property allows the results to be synchronized to another object. For example, if
15 this is set to the main form, parser results will be fired on the application main
16 thread. It may also contain the property “AnalysisOptions AnalysisOptions {get;
17 set;}”, which specifies various criteria that may be used for the analysis process.
18 These criteria may include, for example, enabling text recognition, enabling the
19 use of tables, enabling the use of lists, enabling the use of annotations, and
20 enabling the use of connectors and containers.

21 The Ink Analyzer class will include various methods. For example, this
22 class may include a method entitled “AnalysisRegion Analyze ().” This method
23 starts a synchronous analysis process. Document element data is passed into this
24 method, which describes the current state of the document and indicates what ink
25 in the document needs to be analyzed. With some embodiments, the document

1 element data can be provided as the analysis context object (*i.e.*, AnalysisRegion
2 Analyze (Analysis Context)), as noted above. Alternately, individual ink strokes
3 may be passed to the analysis process, either using a reference to the strokes (*i.e.*,
4 AnalysisRegion Analyze (Strokes)) or referenced as a property of the Ink
5 Analyzer object (*e.g.*, InkAnalyzer.Strokes {get;set}) with no properties passed to
6 the Analyze method.

7 Once the analysis process is complete, this method will return a reference
8 to the document independent analysis context object that has been modified to
9 contain the results of the analysis process. The method also returns an
10 AnalysisRegion value describing the area in the document where results have been
11 calculated.

12 The Ink Analyzer class may also include a method entitled
13 “AnalysisRegion Analyze(AnalysisContext, waitRegion).” This method is the
14 same as the synchronous Analysis Region Analyze () method discussed above, but
15 it only has ink analyzed if results are needed in the specified waitRegion area.
16 More particularly, the call to this method will identify the analysis context object
17 for the document and the region of the analysis context object (referred to as the
18 “waitRegion”) for which the analysis process should analyze synchronously. In
19 certain embodiments, all other regions of the analysis context object will be
20 ignored, unless the analysis process needs to analyze content in those regions in
21 order to complete its analysis of the waitRegion. As discussed above, the analysis
22 context object passed with this method contains a property called the
23 “DirtyRegion” which describes the areas of the document that need analysis. By
24 specifying a particular waitRegion, the software application may obtain the
25 analysis results more quickly for one specific region of interest, rather than having

1 all of the ink data in the document analyzed. When either of these Analyze
2 methods is called, every available analysis process will be executed. Also,
3 because these Analyze methods are synchronous calls, there is no need to execute
4 the reconciliation process upon their conclusion, nor will an event fire once it is
5 complete.

6 The Ink Analyzer class may also include a method entitled
7 “BackgroundAnalyze(AnalysisContext).” This method starts the specified
8 analysis operation, but does so on a separate background analysis thread. Thus,
9 this method will return control to the main processing thread almost immediately
10 while the actual analysis operation completes in the background. In particular, this
11 method will return a value of “true” if the analysis process was successfully
12 started. Again, the AnalysisContext value passed in to the method identifies the
13 analysis context object for the document and indicates what ink in the document
14 needs to be analyzed. Once the analysis operation has completed on the
15 background thread, a Results event is raised to allow the software application to
16 access the results. The event contains the results and the reconcile method which is
17 used to incorporate the results back into the analysis context object for the current
18 state of the document when the results are returned.

19 Each of these three Analyze methods in turn call the method “Clone” in an
20 “Analysis Region” class. Using the “Clone” method, these Analyze methods
21 create the independent document analysis context object that is subsequently
22 modified by the analysis process to show the analysis results.

23 The Ink Analyzer class may also include a method entitled “Reconcile
24 (AnalysisContext current, AnalysisResultsEventArgs resultArgs),” which the
25 software application calls after receiving the results event which was caused by

1 calling the BackgroundAnalyze(AnalysisContext) method. The Reconcile method
2 compares the analysis results contained in the document independent analysis
3 context object with the current version of the analysis context object maintained
4 by the software application. This method identifies the nodes that need to be
5 added and removed from the current version of the analysis context object, and
6 identifies if any of the following properties of an existing node has changed: its
7 recognition results, its location, the ink strokes associated with the node, or any
8 other data associated with the results of the analysis operation. This method also
9 writes these identified changes to the current version of the analysis context
10 object. This method is sensitive to the ordering of context node ordering, such as
11 the order of word context nodes on a line context node.

12 The analysis results (that is, the value of the property
13 AnalysisResultsEventArgs) are passed back with this method, as they contain a
14 public results structure and a private results structure. The public structure is
15 returned so the software application can preview the changes that will occur in the
16 reconcile stage. The private structure is included to prevent the software
17 application from changing any of the analysis results before the reconcile process.

18 The Ink Analyzer class may also include the methods entitled “Recognizers
19 RecognizersPriority()” and “SetHighestPriorityRecognizer(recognizer).” When
20 ink needs to be recognized, the appropriate recognizer will be used based on
21 language and capabilities. Accordingly, the Recognizers RecognizersPriority()
22 method returns the recognition processes in the order in which they will be
23 evaluated by the Ink Analyzer object. The order is determined per system
24 depending upon recognition processes are available, but can be overridden for the
25 software application by calling the SetHighestPriorityRecognizer(recognizer) on

1 the Ink Analyzer object. The InkAnalyzer will enumerate through this ordered list
2 until an appropriate recognizer can be found. The
3 SetHighestPriorityRecognizer(recognizer) method raises the priority of a
4 recognition process. By raising the priority of a particular recognition process,
5 that recognition process will be used if it matches the needed language and
6 capabilities of the current recognition operation. In essence, the
7 SetHighestPriorityRecognizer(recognizer) pushes the designated recognition
8 process to the top of the list returned by the RecognizersPriority method.

9 The Ink Analyzer class may also contain a method entitled
10 "AnalysisRegion Abort()," which may use an analysis context object as a
11 parameter. This method allows for a foreground or background analysis operation
12 to be terminated early. This method returns an analysis region that describes the
13 area that was being analyzed prior to the abort. Thus, if the software application
14 intends to continue the analysis operation at a later time, this region can be merged
15 into the DirtyRegion of the analysis context object for the current state of the
16 document. Still further, the Ink Analyzer class may include an event entitled
17 "AnalysisResultsEventHandler," which fires to the InkAnalyzer object as
18 frequently as practical. More particularly, this event may fire between analysis
19 processes and at least once every five seconds. This event can be used to provide
20 the application with an update as to the status of an ongoing asynchronous analysis
21 process (or processes).

22 The Ink Analysis API may also include classes in addition to the Analysis
23 Context class and the Ink Analyzer class. For example, the Ink Analysis API may
24 include a Context Node class. This class may include various components relating
25 to the context nodes that make up the analysis context object and the document

1 independent analysis context object, such as the property entitled
2 “ContextNodeType Type {get;}.” Each context node has a type, and there is a
3 specific set of rules that each type should adhere to. This includes rules such as
4 what types of child context nodes are allowed and whether or not strokes may be
5 directly associated to the context node or only via its child context nodes.

6 The possible types of context nodes may be defined in a
7 ContextNodeTypes enumeration and may include, for example, the following
8 types: An InkAnnotation node, which represents ink data annotating non-text
9 data; an InkDrawing node which represents ink data forming a drawing, an
10 InkWord node, which represents ink data forming a word, a Line node, which
11 contains one or more InkWord nodes and/or TextWord nodes for words forming a
12 line of text; ListItem node, which may contain Paragraph, Image, or like nodes
13 expected in a list; and a List node, which contains one or more ListItem nodes
14 each describing an entry in the list. The node types may also include a
15 NonInkDrawing node, representing a non-ink drawing image; an Object node,
16 representing data not covered by other values of the ContextNodeType
17 enumeration; a Paragraph node, which contains a one or more Line nodes
18 corresponding to lines forming a paragraph; a Picture or Image node, representing
19 a picture image; a Root node, which serves as the topmost node in an analysis
20 context object; a Table node, which contains nodes representing items making up
21 a table; a TextBox node, representing a text box; a TextWord node; and an
22 UnclassifiedInk node, corresponding to ink data that has not yet been classified.
23 The node types may also include a Group node, for groups of other nodes, an
24 InkBullet node for bullet items, a Row node for documents elements presented in a
25

1 row of a table, and a Cell node for document elements presented in a cell of a
2 table.

3 The Context Node class may also include the property entitled "GUID Id
4 {get;}," which is a globally unique identifier for the current context node. In order
5 to allow access to any desired context node, each context node within a single
6 analysis context object should have unique identifier. This class may also include
7 a property entitled "AnalysisRegion Location {get;}," which identifies the location
8 in the document space where the relevant context node is actually positioned. An
9 AnalysisRegion is a two-dimensional structure grouping one or more possibly
10 disjoint rectangle like structures together. This class may also include the property
11 entitled "StrokeCollection Strokes {get;}", which identifies the ink strokes
12 associated with the relevant context node. With particular embodiments, only leaf
13 context nodes (such as Word, Drawing and Bullet nodes) are allowed by the Ink
14 Analysis API to have strokes. The software application may use this property to
15 reference the strokes at the leaf node level by all ancestor context nodes (e.g., the
16 root node would contain a strokes reference to all strokes in the relevant analysis
17 context object.)

18 Further, this class may include the property entitled "ContextNode
19 ParentNode {get;}," which identifies the parent context node containing the
20 relevant context node. In particular embodiments, context nodes are always
21 created to depend from a parent context node, with the Root context node being a
22 static member of an analysis context object. This class may also include the
23 property "ContextNode[] SubNodes {get;}" which identifies all context nodes that
24 are direct children of the relevant context node. That is, this property will only
25 identify those children context nodes down one level in the analysis context

1 object. For example, the value of this property for a Paragraph context node will
2 only identify the line context nodes contained by the Paragraph node, and not the
3 word context nodes that are children of the line context node.

4 This class may also include the property entitled "RecognitionResult
5 RecognitionResult {get;}," which provides the recognition result as calculated by
6 the relevant recognition analysis process or processes, as the RecognitionResult
7 can represent more than one line of text from in more than one language. The
8 Recognition Result is available for every context node in the document
9 independent analysis context object even though the RecognitionData property
10 which is set by the recognition analysis process and is used to create the
11 RecognitionResult object might only be set at one level of the context node tree to
12 avoid duplication of data. If the node does not have a RecognitionData associated
13 with it, it will either merge the recognition results of all of its subnodes or extract
14 the recognition result from its parent. This class may also include the property
15 entitled "Stream RecognitionData {get; set;}," which is the persistent form of the
16 RecognitionResult value. Again, the recognition analysis process produces the
17 Stream RecognitionData value that is set on the relevant context node. The
18 RecognitionResult object is then constructed based on this value.

19 The Context Node class may further include a property entitled
20 "ContextLink[] Links {get;}," which provides an array of ContextLink objects. A
21 ContextLink object describes an alternative relationship between two context
22 nodes. While context nodes typically have a parent-child relationship with other
23 context nodes, a ContextLink allows for an alternative relationship between
24 context nodes. For example, a ContextLink may allow for a connection between
25 two context nodes, anchoring of one context node to another context node,

1 containment of one context node by another context node, or a desired type of link
2 defined by the software application. ContextLinks may be added to this array by
3 calling the AddLink method. Similarly, ContextLinks may be removed from this
4 array by calling the DeleteLink method.

5 Still further, this class may include the properties “IsContainer {get;}” and
6 “IsInkLeaf {get;}.” The property IsContainer {get;} has the value of “true” if the
7 relevant context node is not a leaf context node (that is, if the relevant context
8 node contains children context nodes and is thus considered a container context
9 node), and has the value of “false” otherwise. The IsInkLeaf {get;} property has
10 the value of “true” if the current context node is not a container context node, and
11 has a value of “false” otherwise. That is, if the current context node does not
12 contain any children context nodes, it is considered a leaf context node. In certain
13 embodiments, an InkLeaf context node is expected to contain references to stroke
14 data, whereas container context nodes do not have this restriction. Container
15 context nodes may or may not reference stroke data, as designated by the software
16 application.

17 The Context Node class may also contain the property “Rect
18 RotatedBoundingBox {get; set;}.” The value of this property is calculated by a
19 layout and classification analysis process. If the ink data associated with the
20 relevant context node is written at an angle, then the bounds for the context node
21 will still be horizontally aligned. The value of the RotatedBoundingBox property,
22 however, will be aligned to the angle at which the ink data associated with the
23 relevant context node was written. Still further, this class may include the
24 property “ReClassifiable {get;},” which informs the InkAnalyzer if it is allowed to
25 modify the values of the relevant context node.

1 In addition to these properties, the Context Node class may also include
2 various methods. For example, this class may include a method entitled
3 “ContextNode CreateSubNode(ContextNodeType type).” This method allows the
4 creation of a child context node of a particular type. In one embodiment, this
5 method may only allow valid child types of the relevant context node to be
6 created, thereby preventing malformed data structures from being created. For
7 example, this method may only allow a Line context node to create InkWord and
8 TextWord child context nodes. This class may also contain a method entitled
9 “void DeleteSubNode(ContextNode node),” which deletes the referenced child
10 context node from the relevant analysis context object. However, in certain
11 embodiments, if the referenced context node still contains strokes or child context
12 nodes, then this method should fail. Also, if the reference context node is not a
13 direct child of the relevant context node, then this method should fail. If a
14 software application implements its own analysis context object and in turn
15 employs this method, it does not delete non-empty context nodes or context nodes
16 that are not direct children of the relevant context node to prevent malformed data
17 structures in the analysis context object.

18 Additionally, this class may include the method “ContextNode[]
19 HitTestSubNodes(AnalysisRegion region),” which returns an array of context
20 node that are located in the specified region. However, only the immediate
21 children nodes of this element are returned, not all descendants. The region is
22 defined by the AnalysisRegion object, which may be a collection of one or more
23 rectangles. In particular embodiments, if any part of a context node’s location
24 intersects the specified region, then that context node will be returned in the array.
25 This method is employed to, for example, create the document independent

1 analysis context object and to reconcile the analysis results with the analysis
2 context object corresponding to the current state of the document. Thus, this
3 method is frequently called, and should be optimized for fast repeated access by
4 the InkAnalyzer object.

5 The Context Node class may also contain a method entitled
6 "MoveStroke(Stroke stroke, ContextNode destination)." This method moves the
7 association of a stroke from one leaf context node to another. In certain
8 embodiments, this method is only used between leaf context nodes. It may also
9 include a method entitled "MoveSubNodeToPosition(int OldIndex, int
10 NewIndex)," which reorders the relevant context node with respect to its sibling
11 context nodes. For example, if the document has three words on a line, *e.g.*, word
12 1, word 2 and word3, then their order is implied by the array of subnodes returned
13 from the parent context node. This method allows their order to be changed, so
14 that, relative to the relevant parent context node, word.1 is specified to be the last
15 word on the line by moving the context node for word 1 from position one to
16 position three.

17 Still further, this class may include the method entitled
18 "AddLink(ContextLink link)," which adds a new ContextLink object to the
19 current context node. In particular embodiments, the ContextLink object should
20 contain a reference to the relevant context node in order for the ContextLink to be
21 successfully added to the array of ContextLinks associated to the relevant context
22 node. It may also contain the method entitled "DeleteLink(ContextLink link)."
23 This method deletes or removes the specified ContextLink object from the array of
24 ContextLinks for the relevant context node. In one embodiment, this method call
25

1 always completes successfully, even if the ContextLink does not exist in the array
2 of ContextLinks associated with the relevant context node.

3 The Ink Analysis API may also include an Analysis Hint class. As with
4 many of the previously described classes, the Analysis Hint class may include a
5 constructor, entitled “AnalysisHint(),” which initializes an Analysis Hint object to
6 an empty state. This class may also include a number of properties, including a
7 property entitled “AnalysisRegion Location {get;}.” This property specifies the
8 location in the document (as an AnalysisRegion) to which the AnalysisHint is
9 applicable. For example, if the document is a free form note with a title section at
10 the top of the page, then the application could set an AnalysisHint for the title
11 region to specify that a horizontal line of ink is expected in that region. This
12 Analysis Hint will help to increase the accuracy of an analysis process.

13 This class may also include a property entitled “string Factoid {get; set;},”
14 which specifies a particular “factoid” that is to be used for the location in the
15 document to which the AnalysisHint is applicable. Factoids provide hints to a
16 recognition process as to an expected use of ink data (*e.g.*, as regular text, digits,
17 postal codes, filenames, and web URLs). This class may also include the
18 properties entitled “RecognizerGuide Guide {get; set;}” and “OverrideLanguageId
19 {get; set;}.” The RecognizerGuide Guide {get; set;} property specifies the writing
20 guide that is to be applied to the location in the document to which the
21 AnalysisHint is applicable. Writing guides may, for example, help improve the
22 accuracy of a recognizer analysis process by specifying to the user and informing
23 the recognizer analysis process where the user will write lines or characters. The
24 OverrideLanguageId {get; set;} property specifies a Language Hint for the
25 document to which the AnalysisHint is applicable. Setting a Language Hint

1 causes the InkAnalyzer object to use the designated language instead of the
2 language specified on the context node.

3 This class may also include a property entitled "PrefixText {get; set;},"
4 which specifies the text that is written or typed prior to a line of ink that is to be
5 recognized. Still further, this class may include a property entitled
6 "RecognitionModes RecognitionFlags {get; set;}," which specifies particular type
7 of modes a recognition process should respect at the location in the document to
8 which the AnalysisHint is applicable. Additionally, this class may include a
9 property entitled "SuffixText {get; set;}," which specifies the text that is written or
10 typed after to a line of ink that is to be recognized, and a property entitled
11 "WordList WordList {get; set;}," which specifies a particular set of words that
12 should be used by a recognition analysis process. Word lists may be used when
13 the expected recognition results are known before the user has actually written
14 input data, such as a list of medical terms that are expected to be written inside a
15 medical form.

16 Still further, this class may include a property entitled "WordMode {get;
17 set;}." If this value is "true," then the analysis process will bias itself to return a
18 single word for the entire analysis region. It may also include a property entitled
19 "Coerce {get; set;}," which, if "true," will force the analysis process to confine its
20 result to any factoid or wordlist value set in the relevant hint. This class may also
21 include a property entitled "AllowPartialDictionaryTerms {get; set;}." If this
22 property value is "true," then the recognition analysis process will be allowed to
23 return partial words from its recognition dictionary.

24 In particular embodiments, the Ink Analysis API may further include an
25 Analysis Region class. This class may include, for example, multiple constructors

1 for constructing an AnalysisRegion object. For example, it may contain a first
2 constructor for constructing an AnalysisRegion object having any region, a second
3 constructor for constructing an AnalysisRegion object based upon the parameters
4 for a two-dimensional rectangle, and a third constructor for constructing an
5 AnalysisRegion object based upon four spatial coordinates. The default
6 constructor may, for example, create an empty region. This class may also include
7 a number of properties. For example, this class may include a property entitled
8 "Rectangle Bounds {get;}," which retrieves the bounding rectangle for the
9 AnalysisRegion, a property entitled "IsEmpty {get;}," which indicates whether the
10 relevant AnalysisRegion object has an empty interior, and a property entitled
11 "IsInfinite {get;}," which indicates whether the relevant AnalysisRegion is set to
12 infinite or not.

13 This class may also include a number of methods, such as a method entitled
14 "AnalysisRegion Clone()," which clones the relevant AnalysisRegion object. This
15 class may also include a method entitled "Equals(AnalysisRegion otherRegion),"
16 which tests whether the specified AnalysisRegion object (referred to as the
17 otherRegion) is identical to the relevant AnalysisRegion object. This method
18 returns a value of "true" if the interior of the specified Analysis Region object is
19 identical to the interior of the relevant Analysis Region object, and otherwise
20 returns a value of "false."

21 This class may further include a method "Intersect(AnalysisRegion
22 regionToIntersect)," which crops down the relevant AnalysisRegion object to the
23 specified analysis region. Thus, the resulting AnalysisRegion object will only
24 include areas that overlapped or intersected the specified analysis region. This
25 class may also include the method entitled "Intersect(Rectangle rectangle)," which

1 crops down the relevant AnalysisRegion object to the specified rectangle. Again,
2 the resulting AnalysisRegion object will only include areas that overlapped or
3 intersected the specified rectangle. It may also include the method entitled
4 “MakeEmpty(),” which initializes the relevant AnalysisRegion object to an empty
5 interior, and the method entitled “MakeInfinite(),” which sets the area occupied by
6 the relevant AnalysisRegion to be infinite. It may further include various methods
7 for uniting or separating differently defined areas, such as method entitled
8 “Union(AnalysisRegion regionToUnion),” which specifies an AnalysisRegion
9 object to union or add to the relevant AnalysisRegion object, and the method
10 entitled “Union(Rectangle rectangle),” which unions a specified rectangle to the
11 relevant Analysis Region object. With this method, the rectangle may be specified
12 in terms of the coordinate space for the relevant AnalysisRegion object. Of
13 course, this class may include numerous other methods for combining areas or
14 extracting one area from another based upon any desired definition for the areas.

15 The Ink Analysis API may also have a Recognition Result class. As with
16 many of the previously discussed classes, the Recognition Result class may
17 include one or more constructors. For example, this class may include a
18 constructor entitled “RecognitionResult(Stream lattice)”, which constructs a
19 RecognitionResults object from a given recognition lattice. In particular
20 embodiments, a recognition lattice is a serialized form of the results from a
21 recognition process. This method may, for example, specify a recognition lattice
22 as a byte array to be used for the construction of the relevant RecognitionResult
23 object. It may also include a constructor entitled
24 “RecognitionResult(ContextNode node),” which constructs a RecognitionResults
25 object from a given context node. It may also include a constructor entitled

1 “RecognitionResult(string Text, int StrokeCount),” which constructs a
2 RecognitionResults object from a specified text value, which in turn is associated
3 to a specified number of strokes and might be used for correction if the recognition
4 process did not come up with an alternate recognition value corresponding to the
5 actual handwritten ink data. Still further, this class may include a constructor
6 entitled “RecognitionResult(RognitionResult leftRecognitionResult,
7 RecognitionResult rightRecognitionResult),” which constructs a
8 RecognitionResults object by merging two existing Recognition Results objects
9 together.

10 The Recognition Result class may also include one or more properties, such
11 as a property entitled “”StrokeCollection StrokeCollection {get;},” which provides
12 an array of stroke indexes representing a collection of strokes that are contained in
13 a single ink object, and a property entitled “RecognitionAlternate TopAlternate
14 {get;},” which provides the best alternate of the recognition result. This class may
15 also include the property entitled “RecognitionConfidence RecognitionConfidence
16 {get;},” which provides the level of confidence (*e.g.*, strong, intermediate, or poor)
17 of the top alternate selection for the current results from a recognition analysis
18 process, and a property entitled “string TopString {get;},” which returns the best
19 result string of the analysis results from a recognition analysis process.

20 The Recognition Results class may also include a number of methods, such
21 as a method entitled “public RecognitionAlternateCollection
22 GetAlternateCollectionFromSelection (selectionStart, selectionLength,
23 maximumAlternates),” which specifies a collection of alternates from a selection
24 within the best result string of analysis results from a recognition analysis process,
25 where each alternate corresponds to only one segment of ink. The input

1 parameters for this method may include, for example, a value which specifies the
2 start of the text selection from which the collection of alternates is returned, a
3 value that specifies the length of the text selection from which the collection of
4 alternates is returned, and a value that specifies the maximum number of alternates
5 to return. This method may then return the RecognitionAlternateCollection
6 collection of alternates from a selection within the best result string of the
7 recognition result.

8 The Recognition Results class may further include a method entitled
9 "RecognitionResult Merge(RecognitionResult left, string separator,
10 RecognitionResult right)." This method may be used to create a new
11 RecognitionResult object from a single string, resulting in a flat lattice, append a
12 single string to the beginning or end of an existing RecognitionResult object, or
13 concatenate a single string in between two existing RecognitionResult objects.
14 This class may also include a method entitled
15 "ModifyTopAlternate(RecognitionAlternate alternate)," which specifies the
16 recognition result to be modified with a known alternate. With some
17 embodiments, by default the best result string of the results of a recognition
18 analysis process corresponds to the top alternate. However, this method can be
19 used to specify that alternates other than the top alternate are used in the results of
20 the recognition analysis process. If the new top alternate results in a different
21 segmentation than the previous one, the ModifyTopAlternate method will
22 automatically update the context nodes to reflect the changes. To retrieve the
23 alternates that can be used to modify the recognition result, this method calls the
24 GetAlternatesFromSelection method. This class may also have a method entitled
25 "Stream Save()," which persistently maintains the relevant RecognitionResults

1 object in the form of a recognition lattice. A recognition lattice is a serialized
2 format used to express the results from a recognition process.

3 The Ink Analysis API may also have an Analysis Options enumerated type.
4 This type may contain one or more fields that specify how ink data will be
5 analyzed by an analysis process, such a field entitled "const AnalysisOptions
6 Default," which enables all available options for the analysis process. This field
7 may, for example, enable text recognition, the use of tables, the use of lists, the use
8 of annotations, the use of connectors and containers, and the use of intermediate
9 results. This type may also include a field entitled "const AnalysisOptions
10 EnableAnnotations," which enables and disables the detection of annotations, a
11 field entitled "const AnalysisOptions EnableConnectorsAndContainers," which
12 enables and disables the detection of connectors and containers, and a field
13 entitled "const AnalysisOptions EnableIntermediateResults," enables and disables
14 the return of analysis results to the software application between the use of
15 different, sequential analysis processes (*e.g.*, between a parsing process and a
16 subsequent recognition process). This type may also have a field entitled "const
17 AnalysisOptions EnableLists," which enables and disables the detection of lists,
18 and a field entitled "const AnalysisOptions EnableTables," which enables and
19 disables the detection of tables. This enumerated type may further include a field
20 entitled "const AnalysisOptions EnableTextRecognition," which enables and
21 disables a text recognition analysis process. However, if additional analysis
22 processes are available (or different versions of the same analysis process), then
23 this type may include additional AnalysisOptions accordingly.

24 Still further, the Ink Analysis API may include an
25 AnalysisResultsEventArgs class. This class may have a constructor entitled

1 “public AnalysisResultsEventArgs(),” which creates a data structure that contains
2 the analysis results and is returned to the software application when the
3 AnalysisResults event is raised. This class may also include a property entitled
4 “InkAnalyzer InkAnalyzer {get;},” which identifies the InkAnalyzer object that
5 performed the analysis process.

6 The API may also have a Line class, which may be useful with some types
7 of operating systems which recognize the use of a “Line” object representing a
8 geometric line. This class may include a constructor, such as a constructor entitled
9 “public Line(Point beginPoint, Point endPoint),” which creates a Line object. This
10 class may also include various properties, such as a property entitled “public Point
11 BeginPoint {get; set;},” which represents the beginning point of the line object
12 and a property entitled “public Point EndPoint {get; set;},” which represents the
13 ending point of the line object.

14 In addition to these classes, the Ink Analysis API may also contain a
15 Recognition Alternate class. This class may include elements representing the
16 possible word matches for segments of ink that are compared to a recognizer’s
17 dictionary. For example, this class may include a property entitled “Line
18 Ascender {get;},” which provides the ascender line of a RecognitionAlternate
19 object that exists on a single line (with a line being represented as two points), a
20 property entitled “public Line Baseline {get;},” which provides the Baseline of a
21 RecognitionAlternate object that exists on a single line, and a property entitled
22 “Line Descender {get;},” which provides the descender line of a
23 RecognitionAlternate object that exists on a single line. This class may also
24 include a property entitled “RecognitionResult Extract {get;},” which provides a
25 RecognitionResults object for the current RecognitionAlternate object. This

1 property can be used, for example, to extract the RecognitionResult object for a
2 word from the RecognitionResult object for a line containing that word.

3 It may also include the property entitled "Line Midline {get;}," which
4 provides the midline for a RecognitionAlternate object that exists on a single line,
5 a property entitled "StrokeCollection Strokes {get;}," which provides the
6 collection of strokes that are contained in an ink object (that is, it provides a
7 StrokeCollection representing the strokes that are associated to the
8 RecognitionResult), and a property entitled "StrokeCollection[] StrokesArray
9 {get;}," which provides a collection of strokes that are contained in one or more
10 ink objects, representing the strokes that are associated with the
11 RecognitionResult. This class also may include a property entitled
12 "RecognitionConfidence RecognitionConfidence {get;}," which provides the level
13 of confidence (*e.g.*, strong, intermediate, or poor) that a recognition analysis
14 process has determined in the recognition of a RecognitionAlternate object or of a
15 gesture. For non-line nodes, the lowest RecognitionConfidence of the children of
16 the relevant context nodes will be returned. It may also contain the property
17 entitled "string RecognizedString {get;}" which specifies the result string of the
18 alternate. Thus, for any context node above a word context node, the results string
19 is concatenated together by this method. For example, a line node will contain a
20 results string that in turn contains the results of all its children or word nodes. A
21 paragraph node will then contain a results string that contains the results of all its
22 children or line nodes.

23 The Recognition Alternate class may also contain one or more methods
24 including, for example, a method entitled "StrokeCollection[]
25 GetStrokesArrayFromTextRange(int selectionstart, int selectionlength)," which

1 specifies a StrokeCollection from each ink object that corresponds to the known
2 text range. This class may also contain a method entitled "StrokeCollection[]
3 GetStrokesFromStrokesArrayRanges(StrokeCollection[] strokesArray)," which
4 specifies the smallest collection of strokes that contains a known input collection
5 of strokes and for which the recognizer can provide alternates. More particularly,
6 the strokes are returned by an array of ink objects each containing an array of
7 stroke indexes for the collection. The collection of ink strokes returned by this
8 method may match the input collection, or it may be larger if the input collection
9 matches only part of the smallest recognition result that includes all of the input
10 strokes. This class may further include a method entitled "StrokeCollection
11 GetStrokesFromStrokesRanges(StrokeCollection strokes)," which specifies the
12 smallest collection of strokes that contains a known input collection of strokes and
13 for which the recognizer can provide alternates, and a method entitled
14 "StrokeCollection GetStrokesFromTextRange(int selectionstart, int
15 selectionlength)," which specifies the StrokeCollection that corresponds to the
16 known text range.

17 This class may further include a method entitled "void
18 GetTextRangeFromStrokes(ref int selectionstart, ref int selectionend,
19 StrokeCollection strokes)," which specifies the smallest range of recognized text
20 for which the recognizer can return an alternate that contains a known set of
21 strokes, and a method entitled "void GetTextRangeFromStrokesArray(ref int
22 selectionstart, ref int selectionend, StrokeCollection[] strokesarray)," which
23 specifies the smallest range of recognized text for which the recognizer can return
24 an alternate that contains a known set of strokes. It also may have a method
25 entitled "RecognitionAlternateCollection SplitWithConstantPropertyValue(GUID

propertyType),” which returns a collection of alternates, which are a division of the alternate on which this method is called. Each alternate in the collection contains adjacent recognition segments which have the same property value for the property passed into the method. For example, this method can be used to obtain alternates that divide an original alternate by level of confidence boundaries (strong, intermediate, or poor) in the recognition result, line boundaries, or segment boundaries. It may further include a method entitled “byte[] GetPropertyValue(GUID propertyType),” which specifies the value of a known property of the alternate, such as the recognizer’s confidence in the alternate. Not all recognition analysis processes will provide a value for all property types, however. Thus, this method provides the data for the types supported by the relevant recognition analysis process.

The Ink Analysis API may also include a Recognition Alternate Collection class. Like many of the classes discussed above, this class may include a constructor, entitled “RecognitionAlternateCollection(),” for creating a RecognitionAlternateCollection object. This class may also include a number of properties, such as a property entitled “Count {get;},” which provides the number of objects or collections contained in a collection of alternate recognition values, a property entitled “IsSynchronized {get;},” which provides a value indicating whether access to the collection of alternate recognition values is synchronized with the software application (*i.e.*, “thread safe”), and a property entitled “SyncRoot {get;},” which provides the object that can be used to synchronize access to the collection of alternate recognition values.

This class may also contain one or more methods, such as a method entitled “virtual void CopyTo(Array array, int index),” which copies all of the elements of

1 the current collection of alternate recognition values to the specified one-
2 dimensional array, starting at the specified destination array index, and a method
3 entitled "IEnumerator IEnumerable.GetEnumerator()," which is a standard
4 implementation of IEnumerable that enables callers to use the for each construct to
5 enumerate through each RecognitionAlternate in the collection of alternate
6 recognition values. This class may also include a method entitled
7 "RecognitionAlternateCollectionEnumerator GetEnumerator()," which returns a
8 RecognitionAlternateCollectionEnumerator that contains all of the objects within
9 the collection of recognition alternate values. This method may be used, for
10 example, to retrieve each object in the collection of recognition alternate values.

11 The Ink Analysis API may additionally include a Recognition Confidence
12 enumeration and a Recognition Mode enumeration, each of which may contain
13 one or more fields relating to a recognition analysis process. For example, the
14 Recognition Confidence class may contain multiple fields, such as a field entitled
15 "Intermediate," indicating that the recognition analysis process is confident that
16 the correct result is in the list of provided alternate recognition values, a field
17 entitled "Poor," which indicates that the recognition analysis is not confident that
18 the result is in the list of provided alternate recognition values, and a field entitled
19 "Strong," which indicates that the recognition analysis process is confident that the
20 best alternate in the alternate recognition values is correct.

21 Similarly, the Recognition Mode class may include fields that that specify
22 how a recognition analysis process interprets electronic ink data and thus
23 determines a recognition result string. For example, this class may include a field
24 entitled "Coerce," which specifies that the recognition analysis process coerce a
25 recognition result based on a factoid that was specified for the context, and a field

1 entitled "Line," which specifies that the recognition analysis process treat the
2 electronic ink data as a single line. This class also may include a field entitled
3 "None," which specifies that the recognition analysis process apply no recognition
4 modes, and a field entitled "Segment," which specifies that the recognition
5 analysis process treat the electronic ink data as forming a single word or character.
6 Still further this class may include a field entitled "TopInkBreaksOnly," which
7 disables multiple segmentation.

8 Still further, the Ink Analysis API may include a Context Link class, which
9 defines how two context nodes may be linked together. The ContextLink object
10 by itself represents which two context nodes are linked, the direction of the link,
11 and the type of link. This class may include a property entitled "ContextNode
12 SourceNode{get;}," which specifies the source context node that is being linked
13 from another context node, a property entitled "ContextLinkType
14 LinkType{get;}," which specifies the type of link relationship that exists between
15 the source and destination context nodes, and a property entitled
16 "CustomLinkType{get;}," which specifies that a custom link is being used. This
17 situation would occur when an application decides to use the linking system of the
18 Ink Analyzer API to represent application specific links beyond what the API can
19 recognize.

20 This class may also include a property entitled "ContextNode
21 DestinationNode {get;}," which specifies the destination context node that is
22 being linked from another context node. There may be two constructors available
23 to this class, which create a relationship between existing source and destination
24 context nodes.

1 This class may also include an enumeration entitled “ContextLinkType
2 enum,” which defines the type of relationship shared by two context nodes. These
3 various link types may include, for example, an “AnchorsTo” type, which
4 describes that one node is anchored to the other node. Both nodes can use the
5 SourceNode or DestinationNode property based on the parsing situation. The link
6 types may also include the type “Contains,” which describes that the one node
7 contains the other node. With this relationship, the container node could be
8 referenced as the SourceNode, while the containee node could be referenced as the
9 DestinationNode. The link types may further include a “PointsTo” type, which
10 describes that one node is pointing to another node. For this relationship, the node
11 doing the pointing could be referenced as the SourceNode, while the node being
12 pointed to could be referenced as the DestinationNode. Still further, the link types
13 may have a “PointsFrom” type, which describes that one node is pointing from the
14 other node. In this relationship, the node pointing away from the other node could
15 be referenced as the SourceNode, while the node being pointed from could be
16 referenced as the DestinationNode.

17 The link types may additionally include a “SpansHorizontally” type, which
18 describes that one node runs the length horizontally of another node, and a
19 “SpansVertically” type, which describes that one node runs the length vertically of
20 another node. For these types, the node covering (strike out, underline, margin
21 bar) the other node, usually written last, could be referenced as the SourceNode,
22 while the node being spanned could be referenced as the DestinationNode. The
23 link types may also include a “Custom” type, which describes that a custom link
24 type has been used. When this value is used, the “CustomLinkType” property on
25 the ContextLink object could provide more details as to the purpose of this link.

Application Model

A Windows Client integrates characteristics of the Web with characteristics of traditional desktop applications. The Application Model provides a framework for secure applications and simplifies development, deployment and maintenance of client applications. This framework provides a simple and consistent user experience. For example, local applications can leverage familiar browser-like features regardless of whether the application is hosted in the browser or is a standalone application, while retaining the benefits of executing on the local client. This framework allows users to leverage their familiarity with the Web, thereby increasing the user's comfort level and reducing the time required to learn to use a new application. The Application Model is part of the System.Windows namespace.

Applications utilizing the Application Model operate in a manner similar to Web pages. When a user browses to an application, the application is automatically installed without requiring user confirmation of the installation, rebooting the client system, or risking the malfunctioning of other applications. In one embodiment, applications download progressively, thereby providing an initial level of interactivity before the application is completely downloaded. Application updates are handled automatically and in a manner that is transparent to the user. Thus, the user always has access to the latest version of the application without having to explicitly perform an application upgrade.

Applications that use the Application Model run locally on the client system regardless of whether the client system is on-line (i.e., actively coupled to the Web) or off-line (i.e., not actively coupled to the Web). This allows an

1 application to provide better performance than a server-based application that
2 needs an active Web connection and continually exchanges data with the server
3 across the Web. After an application is installed on a client system, the application
4 can be accessed from a “Start” menu (like a traditional desktop application) or by
5 navigating to the application (like a Web application). The Application Model
6 contains three primary parts: application lifecycle management, an application
7 framework and a navigation framework.

8 Two different types of applications are supported by the Application Model:
9 an “on-line application” and a “managed application”. Applications utilizing the
10 Application Model can execute in the browser or in a standalone top-level
11 window. An “on-line application” is an application that executes from a server
12 and is hosted in a browser. The application can be cached for offline access or the
13 application may require certain on-line resources to execute properly. A “managed
14 application” is available off-line and is installed on the client. The operating
15 system services the managed application. An entry for the managed application
16 can be added to the “Start” menu on the client. Applications can be downloaded
17 progressively to allow the user to begin interacting with the application as it is
18 being downloaded rather than delaying interaction until an installation process has
19 finished.

20 Applications have an associated application manifest that describes the
21 application’s dependencies, such as additional libraries and resources needed to
22 execute the application. An installer uses the application manifest to control
23 downloading and installation of the application. A “trust manager” is invoked as
24 part of the installation process. The trust manager uses the application manifest to
25 determine what permissions are needed for the application to execute. The

1 application manifest also specifies shell information, such as file associations and
2 whether to add an entry to the Start menu as well as the icon and text for the entry.

3 Applications utilizing the Application Model include markup, code,
4 resources and a manifest. An application is defined and scoped by its application
5 object, which is a global object that persists in memory for the lifetime of each
6 application session. The application object has knowledge of all the resources that
7 belong to the application and provides a boundary between itself and other
8 applications or external resources. The application framework uses the application
9 object to identify, reference and communicate with the application. The
10 application object is also used within the application to manage windows and
11 resources, specify startup and shutdown behavior, handle configuration settings,
12 specify visual styles for the application, share code, state and resources across
13 navigations, and handle application-wide events.

14 A navigation framework supports navigation-based applications that
15 leverage users' familiarity with navigation and journaling activities on the Web to
16 provide a more familiar, consistent user experience on the client system, regardless
17 of whether the application is hosted in the system browser or in a standalone top-
18 level window. Journaling is the process used by the navigation framework to track
19 navigation history. The journal allows users to retrace their steps backward and
20 forward in a linear navigation sequence. Whether a navigation experience is
21 hosted in the browser or in a standalone navigation window, each navigation is
22 persisted in the journal and can be revisited in a linear sequence by using
23 "forward" and "back" buttons or by invoking "go forward" and "go back"
24 methods. Each navigation window has an associated journal.

1 A NavigationApplication class simplifies the task of creating navigation-
2 based applications by providing properties and events related to navigation. The
3 NavigationApplication class includes a startup property that specifies the page or
4 element to which the system navigates when the application is first launched. This
5 class also has a properties collection that allows an application developer to share
6 global state information across pages without having to subclass the application,
7 and supports data binding to these properties.

8 9 EXEMPLARY COMPUTING SYSTEM AND ENVIRONMENT

10 Fig. 4 illustrates an example of a suitable computing environment 400
11 within which the programming framework 132 may be implemented (either fully
12 or partially). The computing environment 400 may be utilized in the computer
13 and network architectures described herein.

14 The exemplary computing environment 400 is only one example of a
15 computing environment and is not intended to suggest any limitation as to the
16 scope of use or functionality of the computer and network architectures. Neither
17 should the computing environment 400 be interpreted as having any dependency
18 or requirement relating to any one or combination of components illustrated in the
19 exemplary computing environment 400.

20 The framework 132 may be implemented with numerous other general
21 purpose or special purpose computing system environments or configurations.
22 Examples of well known computing systems, environments, and/or configurations
23 that may be suitable for use include, but are not limited to, personal computers,
24 server computers, multiprocessor systems, microprocessor-based systems, network
25 PCs, minicomputers, mainframe computers, distributed computing environments

1 that include any of the above systems or devices, and so on. Compact or subset
2 versions of the framework may also be implemented in clients of limited
3 resources, such as cellular phones, personal digital assistants, handheld computers,
4 or other communication/computing devices.

5 The framework 132 may be described in the general context of computer-
6 executable instructions, such as program modules, being executed by one or more
7 computers or other devices. Generally, program modules include routines,
8 programs, objects, components, data structures, etc. that perform particular tasks
9 or implement particular abstract data types. The framework 132 may also be
10 practiced in distributed computing environments where tasks are performed by
11 remote processing devices that are linked through a communications network. In
12 a distributed computing environment, program modules may be located in both
13 local and remote computer storage media including memory storage devices.

14 The computing environment 400 includes a general-purpose computing
15 device in the form of a computer 402. The components of computer 402 can
16 include, by are not limited to, one or more processors or processing units 404, a
17 system memory 406, and a system bus 408 that couples various system
18 components including the processor 404 to the system memory 406.

19 The system bus 408 represents one or more of several possible types of bus
20 structures, including a memory bus or memory controller, a peripheral bus, an
21 accelerated graphics port, and a processor or local bus using any of a variety of
22 bus architectures. By way of example, such architectures can include an Industry
23 Standard Architecture (ISA) bus, a Micro Channel Architecture (MCA) bus, an
24 Enhanced ISA (EISA) bus, a Video Electronics Standards Association (VESA)

1 local bus, and a Peripheral Component Interconnects (PCI) bus also known as a
2 Mezzanine bus.

3 Computer 402 typically includes a variety of computer readable media.
4 Such media can be any available media that is accessible by computer 402 and
5 includes both volatile and non-volatile media, removable and non-removable
6 media.

7 The system memory 406 includes computer readable media in the form of
8 volatile memory, such as random access memory (RAM) 410, and/or non-volatile
9 memory, such as read only memory (ROM) 412. A basic input/output system
10 (BIOS) 414, containing the basic routines that help to transfer information
11 between elements within computer 402, such as during start-up, is stored in ROM
12 412. RAM 410 typically contains data and/or program modules that are
13 immediately accessible to and/or presently operated on by the processing unit 404.

14 Computer 402 may also include other removable/non-removable,
15 volatile/non-volatile computer storage media. By way of example, Fig. 4
16 illustrates a hard disk drive 416 for reading from and writing to a non-removable,
17 non-volatile magnetic media (not shown), a magnetic disk drive 418 for reading
18 from and writing to a removable, non-volatile magnetic disk 420 (e.g., a “floppy
19 disk”), and an optical disk drive 422 for reading from and/or writing to a
20 removable, non-volatile optical disk 424 such as a CD-ROM, DVD-ROM, or other
21 optical media. The hard disk drive 416, magnetic disk drive 418, and optical disk
22 drive 422 are each connected to the system bus 408 by one or more data media
23 interfaces 426. Alternatively, the hard disk drive 416, magnetic disk drive 418,
24 and optical disk drive 422 can be connected to the system bus 408 by one or more
25 interfaces (not shown).

1 The disk drives and their associated computer-readable media provide non-
2 volatile storage of computer readable instructions, data structures, program
3 modules, and other data for computer 402. Although the example illustrates a hard
4 disk 416, a removable magnetic disk 420, and a removable optical disk 424, it is to
5 be appreciated that other types of computer readable media which can store data
6 that is accessible by a computer, such as magnetic cassettes or other magnetic
7 storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or
8 other optical storage, random access memories (RAM), read only memories
9 (ROM), electrically erasable programmable read-only memory (EEPROM), and
10 the like, can also be utilized to implement the exemplary computing system and
11 environment.

12 Any number of program modules can be stored on the hard disk 416,
13 magnetic disk 420, optical disk 424, ROM 412, and/or RAM 410, including by
14 way of example, an operating system 426, one or more application programs 428,
15 other program modules 430, and program data 432. Each of the operating system
16 426, one or more application programs 428, other program modules 430, and
17 program data 432 (or some combination thereof) may include elements of the
18 programming framework 132.

19 A user can enter commands and information into computer 402 via input
20 devices such as a keyboard 434 and a pointing device 436 (e.g., a “mouse”).
21 Other input devices 438 (not shown specifically) may include a microphone,
22 joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and
23 other input devices are connected to the processing unit 404 via input/output
24 interfaces 440 that are coupled to the system bus 408, but may be connected by
25

1 other interface and bus structures, such as a parallel port, game port, or a universal
2 serial bus (USB).

3 A monitor 442 or other type of display device can also be connected to the
4 system bus 408 via an interface, such as a video adapter 444. In addition to the
5 monitor 442, other output peripheral devices can include components such as
6 speakers (not shown) and a printer 446 which can be connected to computer 402
7 via the input/output interfaces 440.

8 Computer 402 can operate in a networked environment using logical
9 connections to one or more remote computers, such as a remote computing device
10 448. By way of example, the remote computing device 448 can be a personal
11 computer, portable computer, a server, a router, a network computer, a peer device
12 or other common network node, and so on. The remote computing device 448 is
13 illustrated as a portable computer that can include many or all of the elements and
14 features described herein relative to computer 402.

15 Logical connections between computer 402 and the remote computer 448
16 are depicted as a local area network (LAN) 450 and a general wide area network
17 (WAN) 452. Such networking environments are commonplace in offices,
18 enterprise-wide computer networks, intranets, and the Internet.

19 When implemented in a LAN networking environment, the computer 402 is
20 connected to a local network 450 via a network interface or adapter 454. When
21 implemented in a WAN networking environment, the computer 402 typically
22 includes a modem 456 or other means for establishing communications over the
23 wide network 452. The modem 456, which can be internal or external to computer
24 402, can be connected to the system bus 408 via the input/output interfaces 440 or
25 other appropriate mechanisms. It is to be appreciated that the illustrated network

1 connections are exemplary and that other means of establishing communication
2 link(s) between the computers 402 and 448 can be employed.

3 In a networked environment, such as that illustrated with computing
4 environment 400, program modules depicted relative to the computer 402, or
5 portions thereof, may be stored in a remote memory storage device. By way of
6 example, remote application programs 458 reside on a memory device of remote
7 computer 448. For purposes of illustration, application programs and other
8 executable program components such as the operating system are illustrated herein
9 as discrete blocks, although it is recognized that such programs and components
10 reside at various times in different storage components of the computing device
11 402, and are executed by the data processor(s) of the computer.

12 An implementation of the framework 132, and particularly, the API 142 or
13 calls made to the API 142, may be stored on or transmitted across some form of
14 computer readable media. Computer readable media can be any available media
15 that can be accessed by a computer. By way of example, and not limitation,
16 computer readable media may comprise "computer storage media" and
17 "communications media." "Computer storage media" include volatile and non-
18 volatile, removable and non-removable media implemented in any method or
19 technology for storage of information such as computer readable instructions, data
20 structures, program modules, or other data. Computer storage media includes, but
21 is not limited to, RAM, ROM, EEPROM, flash memory or other memory
22 technology, CD-ROM, digital versatile disks (DVD) or other optical storage,
23 magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage
24 devices, or any other medium which can be used to store the desired information
25 and which can be accessed by a computer.

1 “Communication media” typically embodies computer readable
2 instructions, data structures, program modules, or other data in a modulated data
3 signal, such as carrier wave or other transport mechanism. Communication media
4 also includes any information delivery media. The term “modulated data signal”
5 means a signal that has one or more of its characteristics set or changed in such a
6 manner as to encode information in the signal. By way of example, and not
7 limitation, communication media includes wired media such as a wired network or
8 direct-wired connection, and wireless media such as acoustic, RF, infrared, and
9 other wireless media. Combinations of any of the above are also included within
10 the scope of computer readable media.

11 Alternatively, portions of the framework may be implemented in hardware
12 or a combination of hardware, software, and/or firmware. For example, one or
13 more application specific integrated circuits (ASICs) or programmable logic
14 devices (PLDs) could be designed or programmed to implement one or more
15 portions of the framework.

16 Notionally, a programming interface may be viewed generically, as shown
17 in Fig. 5 or Fig. 6. Fig. 5 illustrates an interface Interface1 as a conduit through
18 which first and second code segments communicate. Fig. 6 illustrates an interface
19 as comprising interface objects I1 and I2 (which may or may not be part of the
20 first and second code segments), which enable first and second code segments of a
21 system to communicate via medium M. In the view of Fig. 6, one may consider
22 interface objects I1 and I2 as separate interfaces of the same system and one may
23 also consider that objects I1 and I2 plus medium M comprise the interface.
24 Although Figs. 5 and 6 show bi-directional flow and interfaces on each side of the
25 flow, certain implementations may only have information flow in one direction (or

1 no information flow as described below) or may only have an interface object on
2 one side. By way of example, and not limitation, terms such as application
3 programming or program interface (API), entry point, method, function,
4 subroutine, remote procedure call, and component object model (COM) interface,
5 are encompassed within the definition of programming interface.

6 Aspects of such a programming interface may include the method whereby
7 the first code segment transmits information (where "information" is used in its
8 broadest sense and includes data, commands, requests, etc.) to the second code
9 segment; the method whereby the second code segment receives the information;
10 and the structure, sequence, syntax, organization, schema, timing and content of
11 the information. In this regard, the underlying transport medium itself may be
12 unimportant to the operation of the interface, whether the medium be wired or
13 wireless, or a combination of both, as long as the information is transported in the
14 manner defined by the interface. In certain situations, information may not be
15 passed in one or both directions in the conventional sense, as the information
16 transfer may be either via another mechanism (e.g. information placed in a buffer,
17 file, etc. separate from information flow between the code segments) or non-
18 existent, as when one code segment simply accesses functionality performed by a
19 second code segment. Any or all of these aspects may be important in a given
20 situation, e.g., depending on whether the code segments are part of a system in a
21 loosely coupled or tightly coupled configuration, and so this list should be
22 considered illustrative and non-limiting.

23 This notion of a programming interface is known to those skilled in the art
24 and is clear from the foregoing detailed description of the invention. There are,
25 however, other ways to implement a programming interface, and, unless expressly

1 excluded, these too are intended to be encompassed by the claims set forth at the
2 end of this specification. Such other ways may appear to be more sophisticated or
3 complex than the simplistic view of Figs. 5 and 6, but they nonetheless perform a
4 similar function to accomplish the same overall result. We will now briefly
5 describe some illustrative alternative implementations of a programming interface.

6 7 A. FACTORING

8 A communication from one code segment to another may be accomplished
9 indirectly by breaking the communication into multiple discrete communications.
10 This is depicted schematically in Figs. 7 and 8. As shown, some interfaces can be
11 described in terms of divisible sets of functionality. Thus, the interface
12 functionality of Figs. 5 and 6 may be factored to achieve the same result, just as
13 one may mathematically provide 24, or 2 times 2 times 3 times 2. Accordingly, as
14 illustrated in Fig. 7, the function provided by interface Interface1 may be
15 subdivided to convert the communications of the interface into multiple interfaces
16 Interface1A, Interface 1B, Interface 1C, etc. while achieving the same result. As
17 illustrated in Fig. 8, the function provided by interface I1 may be subdivided into
18 multiple interfaces I1a, I1b, I1c, etc. while achieving the same result. Similarly,
19 interface I2 of the second code segment which receives information from the first
20 code segment may be factored into multiple interfaces I2a, I2b, I2c, etc. When
21 factoring, the number of interfaces included with the 1st code segment need not
22 match the number of interfaces included with the 2nd code segment. In either of the
23 cases of Figs. 7 and 8, the functional spirit of interfaces Interface1 and I1 remain
24 the same as with Figs. 5 and 6, respectively. The factoring of interfaces may also
25 follow associative, commutative, and other mathematical properties such that the

1 factoring may be difficult to recognize. For instance, ordering of operations may
2 be unimportant, and consequently, a function carried out by an interface may be
3 carried out well in advance of reaching the interface, by another piece of code or
4 interface, or performed by a separate component of the system. Moreover, one of
5 ordinary skill in the programming arts can appreciate that there are a variety of
6 ways of making different function calls that achieve the same result.

8 B. REDEFINITION

9 In some cases, it may be possible to ignore, add or redefine certain aspects
10 (e.g., parameters) of a programming interface while still accomplishing the
11 intended result. This is illustrated in Figs. 9 and 10. For example, assume interface
12 Interface1 of Fig. 5 includes a function call *Square(input, precision, output)*, a call
13 that includes three parameters, *input*, *precision* and *output*, and which is issued
14 from the 1st Code Segment to the 2nd Code Segment., If the middle parameter
15 *precision* is of no concern in a given scenario, as shown in Fig. 9, it could just as
16 well be ignored or even replaced with a *meaningless* (in this situation) parameter.
17 One may also add an *additional* parameter of no concern. In either event, the
18 functionality of square can be achieved, so long as output is returned after input is
19 squared by the second code segment. *Precision* may very well be a meaningful
20 parameter to some downstream or other portion of the computing system;
21 however, once it is recognized that *precision* is not necessary for the narrow
22 purpose of calculating the square, it may be replaced or ignored. For example,
23 instead of passing a valid *precision* value, a meaningless value such as a birth date
24 could be passed without adversely affecting the result. Similarly, as shown in Fig.
25 10, interface I1 is replaced by interface I1', redefined to ignore or add parameters

1 to the interface. Interface I2 may similarly be redefined as interface I2', redefined
2 to ignore unnecessary parameters, or parameters that may be processed elsewhere.
3 The point here is that in some cases a programming interface may include aspects,
4 such as parameters, that are not needed for some purpose, and so they may be
5 ignored or redefined, or processed elsewhere for other purposes.

6 7 C. INLINE CODING

8 It may also be feasible to merge some or all of the functionality of two
9 separate code modules such that the "interface" between them changes form. For
10 example, the functionality of Figs. 5 and 6 may be converted to the functionality
11 of Figs. 11 and 12, respectively. In Fig. 11, the previous 1st and 2nd Code Segments
12 of Fig. 5 are merged into a module containing both of them. In this case, the code
13 segments may still be communicating with each other but the interface may be
14 adapted to a form which is more suitable to the single module. Thus, for example,
15 formal Call and Return statements may no longer be necessary, but similar
16 processing or response(s) pursuant to interface Interface1 may still be in effect.
17 Similarly, shown in Fig. 12, part (or all) of interface I2 from Fig. 6 may be written
18 inline into interface I1 to form interface I1". As illustrated, interface I2 is divided
19 into I2a and I2b, and interface portion I2a has been coded in-line with interface I1
20 to form interface I1". For a concrete example, consider that the interface I1 from
21 Fig. 6 performs a function call *square (input, output)*, which is received by
22 interface I2, which after processing the value passed with *input* (to square it) by
23 the second code segment, passes back the squared result with *output*. In such a
24 case, the processing performed by the second code segment (squaring *input*) can
25 be performed by the first code segment without a call to the interface.

D. DIVORCE

A communication from one code segment to another may be accomplished indirectly by breaking the communication into multiple discrete communications. This is depicted schematically in Figs. 13 and 14. As shown in Fig. 13, one or more piece(s) of middleware (Divorce Interface(s), since they divorce functionality and / or interface functions from the original interface) are provided to convert the communications on the first interface, Interface1, to conform them to a different interface, in this case interfaces Interface2A, Interface2B and Interface2C. This might be done, e.g., where there is an installed base of applications designed to communicate with, say, an operating system in accordance with an Interface1 protocol, but then the operating system is changed to use a different interface, in this case interfaces Interface2A, Interface2B and Interface2C. The point is that the original interface used by the 2nd Code Segment is changed such that it is no longer compatible with the interface used by the 1st Code Segment, and so an intermediary is used to make the old and new interfaces compatible. Similarly, as shown in Fig. 14, a third code segment can be introduced with divorce interface DI1 to receive the communications from interface I1 and with divorce interface DI2 to transmit the interface functionality to, for example, interfaces I2a and I2b, redesigned to work with DI2, but to provide the same functional result. Similarly, DI1 and DI2 may work together to translate the functionality of interfaces I1 and I2 of Fig. 6 to a new operating system, while providing the same or similar functional result.

E. REWRITING

1 Yet another possible variant is to dynamically rewrite the code to replace
2 the interface functionality with something else but which achieves the same
3 overall result. For example, there may be a system in which a code segment
4 presented in an intermediate language (e.g. Microsoft IL, Java ByteCode, etc.) is
5 provided to a Just-in-Time (JIT) compiler or interpreter in an execution
6 environment (such as that provided by the .Net framework, the Java runtime
7 environment, or other similar runtime type environments). The JIT compiler may
8 be written so as to dynamically convert the communications from the 1st Code
9 Segment to the 2nd Code Segment, i.e., to conform them to a different interface as
10 may be required by the 2nd Code Segment (either the original or a different 2nd
11 Code Segment). This is depicted in Figs. 15 and 16. As can be seen in Fig. 15, this
12 approach is similar to the Divorce scenario described above. It might be done, e.g.,
13 where an installed base of applications are designed to communicate with an
14 operating system in accordance with an Interface 1 protocol, but then the operating
15 system is changed to use a different interface. The JIT Compiler could be used to
16 conform the communications on the fly from the installed-base applications to the
17 new interface of the operating system. As depicted in Fig. 16, this approach of
18 dynamically rewriting the interface(s) may be applied to dynamically factor, or
19 otherwise alter the interface(s) as well.

20 It is also noted that the above-described scenarios for achieving the same or
21 similar result as an interface via alternative embodiments may also be combined in
22 various ways, serially and/or in parallel, or with other intervening code. Thus, the
23 alternative embodiments presented above are not mutually exclusive and may be
24 mixed, matched and combined to produce the same or equivalent scenarios to the
25 generic scenarios presented in Figs. 5 and 6. It is also noted that, as with most

1 programming constructs, there are other similar ways of achieving the same or
2 similar functionality of an interface which may not be described herein, but
3 nonetheless are represented by the spirit and scope of the invention, i.e., it is noted
4 that it is at least partly the functionality represented by, and the advantageous
5 results enabled by, an interface that underlie the value of an interface.

6 7 **Conclusion**

8 Although the invention has been described in language specific to structural
9 features and/or methodological acts, it is to be understood that the invention
10 defined in the appended claims is not necessarily limited to the specific features or
11 acts described. Rather, the specific features and acts are disclosed as exemplary
12 forms of implementing the claimed invention.